KONSTRUKČNÍ ELEKTRONIKA A Radio

ROČNÍK VI/2001. ČÍSLO 6

Amatérské RADIO

ŘADA B - PRO KONSTRUKTÉRY

ROČNÍK L/2001. ČÍSLO 6

V TOMTO SEŠITĚ

KONSTRUKČNÍ ELEKTRONIKA A RADIO

Vydavatel: AMARO spol. s r. o.
Redakce: Radlická 2, 150 00 Praha 5, tel.: (02) 57 31 73 11, tel./fax: (02) 57 31 73 10.
Šéfredaktor ing. Josef Kellner, sekretářka redakce Eva Kelárková, tel. 57 31 73 14.
Ročně vychází 6 čísel. Cena výtisku 36 Kč.
Rozšiřuje PNS a. s., Transpress spol. s r. o., Mediaprint & Kapa a soukromí distributoři.
Předplatné v ČR zajišťuje **Amaro** spol. s r. o. - Michaela Jiráčková, Hana Merglová (Radlická 2, 150 00 Praha 5, tel./fax: (02) 57 31 73 13, 57 31 73 12). Distribuci pro předplatitele také provádí v zastoupení vydavatele společnost Předplatné tisku s. r. o., Abocentrum, Moravské náměstí 12D, P. O. BOX 351, 659 51 Brno; tel: (05) 4123 3232; fax: (05) 4161 6160; abocentrum@pns.cz; reklamace - tel.: 0800-171 181.
Objednávky a predplatné v Slovenskej republike vybavuje MAGNET-PRESS Slovakia s. r. o., Teslova 12, P. O. BOX 169, 830 00 Bratislava 3, tel./ /fax (02) 44 45 45 59, (02) 44 45 06 97 - predplatné, (02) 44 45 46 28 - administratíva; email: magnet@press.sk
Podávání novinových zásilek povoleno Českou poštou - ředitelstvím OZ Praha (č.j. nov 6005/96 ze dne 9. 1. 1996).
Inzerci v ČR přijímá redakce, Radlická 2, 150 00 Praha 5, tel.: (02) 57 31 73 11, tel./fax: (02) 57 31 73 10.
Inzerci v SR vyřizuje MAGNET-PRESS Slovakia s. r. o., Teslova 12, 821 02 Bratislava, tel./ /fax (02) 44 45 06 93.
Za původnost a správnost příspěvků odpovídá autor (platí i pro inzerci). Nevyžádané rukopisy nevracíme.
http://www.aradio.cz; *E-mail*: pe@aradio.cz
ISSN 1211-3557, MKČR 7443

Dějiny přenosu zpráv na dálku

Historie elektřiny a magnetizmu

Denes von Mihály

Mezi významné osobnosti, které se zapsaly do dějin rozvoje televizní techniky v jejích počátcích, byl nesporně maďarský fyzik Dénes von Mihály. Narodil se 7. 7. 1894 v Maďarském Gödölö, ale v mládí se především zajímal o automobily. Měl německé vychování, což mu umožnilo studovat od roku 1912 v Mnichově a pak v Budapešti fyziku a elektrotechniku.

Během studií se seznámil s teorií přenosu obrazů na dálku a tak již v roce 1919 v Budapešti předváděl přenos obrazů pomocí svého přístroje, který nazval Telehor. Bylo to na vzdálenost asi 5 km a po vedení. Jeho přístroj pracoval na mechanicko-optickém principu s Nipkowovým kotoučem. Za dalšími pokusy se vydal do Německa, kde v roce 1923 v Berlíně - Wilmersdorfu zřídil televizní laboratoř.

První úspěchy jej dovedly k tomu, že založil společnost s názvem Telehor A. G. Dostavily se i praktické výsledky - objevil princip řádkového rozhladu s vodorovným vychylováním a vychylování rastru pomocí pilovitého napětí. Na tento princip obdržel 25. 12. 1923 v Německu patent. Mihály pak spolupracoval s AEG a říšskými poštami a při svých dalších pokusech již měl podporu „referátu pro bezdrátovou telegrafii a různé úkoly“ říšských pošt a také společnosti TEKADE (Telefonapparate Kabel und Drahtwerke A. G. - známé z různých inkurantních přístrojů a součástek).

Der sprechende Film

Von

Dénes von Mihály

Mit 99 Textfiguren



1 9 2 8

Titulní list Mihályho knihy „Der sprechende Film“, která vyšla roku 1928 v Berlíně. Autor se v ní zabývá přenosem zvuku, obrazu a filmovou technikou

V únoru 1928 předváděl přenosy Diaskopie ze své laboratoře do říšského telegrafního technického úřadu a 11. 5. téhož roku předvedl již zájemcům z řad výrobních firem přenos obrazu o rozměrech 4x4 cm. Celkem třicetiřádkový obraz byl rozložen do 900 bodů (dnešní systémy používají rozklad na asi půl milionu bodů!) a přenášen po telefonním kabelu dlouhém asi 2,5 km. Toto předváděl i na 5. německé výstavě rozhlasu v Berlíně dne 31. srpna 1928. Byla to sice zajímavost, ale kvalita obrazu nemohla diváky příliš nadchnout.

Německé říšské pošty v noci z 8. na 9. března 1929 umožnily v Berlíně pokus s bezdrátovým přenosem obrazu, tehdy bez zvuku, s pomocí Mihályho rozkladového zařízení. Signál byl přenášen pomocí středovlnného vysílače a přijímán ve výzkumných laboratořích velkých berlínských firem, které se zajímaly o televizní techniku. Bohužel, celý pokus byl podřízen technicko-fyzikálním omezením, které u Mihályho systému nemohly být překročeny. Jeho firma Telehor A. G. postupně upadala a zůstala mu jen hrstka přívrženců.

Od roku 1933 se pak věnoval pouze konstrukcím elektrooptických součástek pro televizní techniku. V roce 1935 založil spolu s fyzikem E. H. Traubem firmu Mihály & Traub, která se věnovala převážně výrobě Kerrových článků (speciální kyveta s vhodnou kapalinou umístěná v silném elektrickém poli - využívá se např. k modulaci polarizovaného světla).

Zemřel 29. 8. 1953 v Berlíně. I když se jeho teorie a pokusy dostaly do slepé uličky, přece jen v začátcích televize měla jeho práce velký význam.

Max Dieckmann

Když se dnes díváme na televizní obraz rozložený na 625 řádků, stěží si dokážeme představit, jak asi vypadal televizní obraz přenášený pomocí rozkladu do 20 řádků, jak byl poprvé předveden při přenosu jednoduchých náčrtů v době rodící se televize.

Jedním z vědců, kteří se nejvíce zasloužili o rozvoj televizní techniky

v jejích počátcích, byl německý fyzik, odborník v rozvíjející se rozhlasové a televizní technice, Max Dieckmann. Narodil se 5. července 1882 v městečku Harzdorf Herrmannsacker, studoval na gymnaziu v Lipsku. Následovala pak vysokoškolská studia fyziky v Göttingenu, Lipsku a Strassburgu. Na technické vysoké škole v Mnichově ve věku 23 let svá studia dokončil a Bavorsko se od té doby stalo jeho domovinou. Jako mladý fyzik přednášel a setkal se podobně jako Rosing v Rusku s objevem Braunovy obrazovky, kterou intuitivně považoval za významný stavební prvek, který umožní přenášet obraz na dálku.

Bylo to již v roce 1906, kdy se Dieckmannovi podařilo přenést prvé primitivní a statické obrázky s rozkladem na 20 řádků, jako zdroj signálu tehdy posloužilo zařízení na principu Nipkowova kotouče, ale na místě otvorů byly spirálovitě uspořádány drátěné kartáčky, spínané přes kovové šablony. Po těchto primitivních pokusech za pomoci soukromého kapitálu zřídil v roce 1908 „Bezdrátovou pokusnou stanici pro telegraf a vzdušnou elektřinu" v Gräfelfingu. Tam pak prováděl pokusy se statickou elektřinou, přemýšlel nad otázkami přenosu obrazů, rádiového spojení s letadly a letecké navigace a úspěšně je řešil.

Dalším mezníkem v jeho životě ve vztahu k televizní technice byl v roce 1925 objev principu rozkladové elektronky, na který přišli společně při práci s Rudolfem Hellem. Patent ovšem obdržel později Farnsworth.

V roce 1936 byl Dieckmann povolán na vysokou technickou školu do Mnichova, aby tam vedl institut (dnes bychom řekli katedru) radiotechniky. Po roce se stal vědeckým vedoucím nově založeného leteckého výzkumného ústavu v Ober-pfaffenhofenu, kde strávil zbytek svého aktivního života.

V roce 1947 odešel do penze a žil pak opět v bavorském Gräfelfingu. Tam tento pionýr letecké spojovací techniky a televizní techniky zemřel 28. července 1960.

Vladimir Kosma Zworykin

Dalším technikem, který posunul vývoj v oblasti televizní techniky o stupeň výše, byl Rus žijící v Americe, Wladimir Kosma Zworykin, který se narodil 30. 7. 1889 ve městě Murom v Rusku. Po dokončení studia na Institutu technologie v Petersburgu odešel dále studovat do Francie. Byl studentem i u známého profesora Rosinga.

Během první světové války sloužil u ruského spojovacího vojska a v roce 1919 emigroval do USA. Tam pracoval v Pittsburgu u Westighouse Electric Corporation. Jeho prvním obecně známým úspěchem bylo v roce 1923 získání patentu na použití ikonoskopu ve snímací kameře na plně elektronickém principu, což znatelně zlepšilo kvalitu získávaného obrazu. V roce 1924 získává patent na první plně elektronický televizní systém, kde jako přijímací přístroj byl tzv. kineskop, předchůdce dnešních televizorů. Zworykin pracoval od roku 1929 pro RCA, kde se stal nakonec i ředitelem.

Televizní technika však nebyla jeho jediným oborem, kterému se věnoval - spíše naopak, i když se věnoval i barevné televizi, na kterou získal patent v roce 1928. Zabýval se snímacími elektronkami citlivými na infračervené světlo (vymyslel elektronický systém pro ostřelovače-snipery, použitý nakonec ve 2. světové válce), elektronovými mikroskopy a v nich používanými násobiči elektronů, scintilačními prvky a detektory radiace. Firma RCA byla pro Farnsworthovu společnost silná konkurence. Proto Farnsworth nakonec vytvořil v roce 1949 spolu s dalšími společnostmi jeden z největších koncernů, International Telephone and Telegraph Company - ITT.

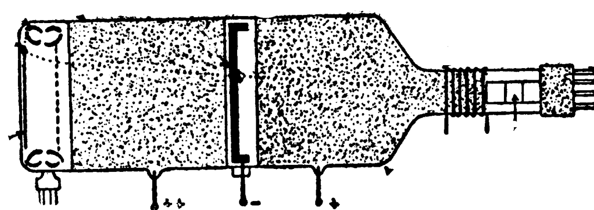
V 50. a 60. létech se Zworykinovi dostalo množství významných ocenění - od jmenování čestným viceprezidentem RCA (1954), po ocenění nejrůznějšími medailemi a čestným členstvím v mnoha organizacích. Stal se např. prvním prezidentem Sdružení pro lékařskou elektroniku a biologické inženýrství, získal Faradayovu medaili udělovanou ve Velké Británii (1965) a byl též autorem řady významných vědeckých publikací. Zemřel den před svými 93. narozeninami, 29. 7. 1982 v Princetonu, USA.

Philo Taylor Farnsworth

Jak již bylo řečeno, jedním z prvních „televizních" průkopníků v Evropě byl J. L. Baird, který pracoval především v oblasti dálkových přenosů obrazu. Další vědci pracovali na opačné straně Atlantiku.

Jedním z nich byl Philo Taylor Farnsworth. Ten se narodil 19. 8. 1906 v Beaveru ve státě Utah a rodina se brzy přestěhovala do Rigby ve státě Idaho, kde mohl Farnsworth junior navštěvovat vyšší školu. Věnoval se hlavně studiu molekulární teorie hmoty, elektronové a Einsteinově teorii.

V roce 1922 si přečetl populárně vědecké pojednání ruského televizního technika Borise Lvoviče Rosinga o „elektrických přenosech jednoduchých obrazů na dálku". Tato myšlenka ho natolik zaujala, že se sám podobným pokusům věnoval.



Nákres Farnsworthovy snímací obrazové elektronky s velkou citlivostí, určené pro špatně osvětlené scény

V letech 1923 až 1925 navštěvoval univerzitu v Provo (Utah), ale ve druhém ročníku mu zemřel otec a univerzitu nedokončil. Podařilo se mu získat peníze na realizaci myšlenky svého vylepšeného televizního systému. Začal se svými výzkumy a brzy spatřila světlo světa jeho elektronická snímací televizní kamera.

Jako předpoklad dalšího rozvoje se mu podařilo vyvinout v roce 1928 elektronickou snímací elektronku zvanou Dissektor, což byl základní prvek jeho nové kamery, která již uměla plošný obraz elektronicky rozčlenit na jednotlivé body. Ve stejném roce obdržel na tento způsob rozkladu obrazu patent, a nové možnosti, které tento princip přinesl, demonstroval na Franklinově institutu ve Filadelfii.

Jeho kamery se vyráběly delší dobu průmyslově v televizních laboratořích v San Franciscu, které byly v roce 1929 přejmenovány na Farnsworth Television Inc. of California. Ještě v roce 1936 německá televizní společnost vyráběla kameru na tomto principu s názvem „olympijská kamera". To již měl za sebou řadu patentů - na způsob rozkladu, na zaostřování, synchronizaci, změnu kontrastu, výkonu atd. I přes nesporný pokrok v kvalitě přenášeného obrazu stále nebylo možné soupeřit se záznamem obrazu na filmový pás.

Ke komerčnímu využití svých vynálezů založil tento nadaný vědec v roce 1932 společnost Philo Corporation ve Fort Wayne (Indiana). Tam se od roku 1938 až do vstupu Spojených států do války vyráběly i televizní přijímače. Během válečných let se věnoval vylepšování tehdejší radarové techniky.

Po válce založil spolu s dalšími společnostmi ohromný koncern ITT. Farnsworth sám pak ještě pracoval v oblasti využití atomové energie a je držitelem několika patentů i z oboru nukleární fúze. Celkem do své smrti 11. 3. 1971 v Salt Lake City získal přes 300 patentů.

Možná více než technici jej znají filatelisté - jeho portrét najdeme i na amerických známkách z roku 1983.

Literatura

[1] *Zworykin, V. K.:* Television in Science and Industry. New York, 1958.

[2] *Mihály, D.*: Der sprechende Film. Berlin, 1928.

[3] *Seger, J.*: Televize - dílo generecí. Praha, 1978.

Ing. Jiří Peček, OK2QX

Ovládání hraček počítačem

Přípravky pro výuku programování

Ing. Michal Černý

Počítače (PC) se dnes používají téměř výhradně uživatelsky, tj. provozují se na nich hotové zakoupené programy, pomocí kterých se vykonává cílová práce - např. editují texty, obrázky apod., vede účetnictví, využívá Internet atd. atd. Vytvořit potřebné programy uživatelé neumějí a ani je to nezajímá. Ukazuje se však, že znalost programování je užitečná, a že programátoři mají vynikající profesionální uplatnění. Programování také v sobě soustřeďuje v daleko vyšší míře než jiné činnosti schopnost analyzovat problémy, formulovat myšlenky a logicky myslet, což všechno jsou dovednosti, které přispívají k rozvoji osobnosti.

Programovat je vhodné se učit již v dětském věku, kdy to nejlépe „leze do hlavy“. Tvorba vlastních programů je pro děti též vhodnou alternativou k pouhému hraní her na PC. Děti je však k programování nutné motivovat, aby jejich zájem rychle neochabl. Výuku programování lze učinit zajímavou např. tím, že programy přes porty PC ovládají a řídí mechanické hračky a vykonávají tak atraktivní činnost, která přitahuje k další práci. Při stavbě hraček si děti také rozvíjejí svoji zručnost v oblasti hardware. Navržené hračky od nejjednodušších až po složité a příklady k nim vytvořených programů (v jazyce Pascal) tvoří náplň tohoto čísla KE. Přitom se předpokládá, že děti nebo jejich blízcí dospělí mají základní znalost programovacího jazyka, vlastní výuka programování není obsahem tohoto článku.

Popisované náměty hraček a programového vybavení najdou samozřejmě uplatnění i ve světě dospělých, kde mohou sloužit jako vodítko a inspirace pro vlastní konstrukce měřicích přípravků a ovládacích zařízení.

Úvod

Ačkoli věkem rozhodně nepatřím ještě mezi pamětníky, v oblasti vývoje výpočetní techniky pamatuji celkem dost. Nechci se však věnovat popisu a historii technického vývoje počítačů, chci jen na jejím pozadí sledovat přístup k nim prostřednictvím několika osobních postřehů. Tyto zážitky mě přesvědčují o smysluplnosti tématu, které otvíráme.

Když jsem se poprvé setkal s počítačem na jakési exkurzi, byl to sál plný kovových skříní, mezi nimiž lehce zmateně pobíhalo asi deset operátorů. Všude hučely ventilátory, občas zarachotila čtečka děrných štítků a zaklepal elektrický psací stroj, který byl jediným výstupním zařízením tohoto monstra. Vysvětlovali nám, tehdy školákům základní devítileté školy, jaký je to úžasný výdobytek socializmu (tak krásné slovo utkví v paměti na dlouhá léta), a že když budeme pilně studovat, můžeme jako tito vědci s podobnými stroji pracovat také.

První programátorské začátky jsem si odbyl na střední škole. Byla to doba, kdy se objevily i první mikroprocesory, kdy v Amatérském radiu sporadicky vycházely první články o amatérských mikropočítačích v USA, a kdy se přes Tuzex daly sehnat první programovatelné kalkulátory. A co se, panečku, všechno vešlo do těch padesáti instrukcí a deseti pamětí na čísla! Kolik hříček i užitečných prográmků tehdy vznikalo pod lavicí.

Ačkoli od toho prvního setkání uplynulo jen pár let, přístup se pronikavě změnil. Ubylo vět o úspěších socializmu, ony by také nad kalkulátorem Ti-58, stejným typem, který se podílel i na letech Američanů na Měsíc, jaksi nevyzněly. Avšak především byla tato technika dostupná i nám, středoškolákům, i když zdaleka ne všem, a jen díky vekslákům, bonům a Tuzexu, a za cenu převyšující měsíční plat rodičů. Nebyla to už technika jen pro vědce v klimatizovaných sálech.

Na střední škole jsem se také poprvé setkal s výukou programování v jazyce, konkrétně v Pascalu. Učili nás lektoři z MFF UK, kde jsme také měli i možnost své programy odladit a spouštět na EC1050 - doufám, že si ten typ pamatuji správně. Práce s děrnými štítky sice nebyla nic moc, ale zato výsledkem už byla tištěná sjetina na papíře, v níž jsme mohli používat i semigrafiku, a to bylo opravdu něco jiného, než malá červená číslíčka na displeji kalkulátorů.

Výuka programování byla koncipovaná jako výuka jazyka i základních algoritmů, a vzpomínám, že to velmi nezáživné a často se nám ztrácel smysl toho, co právě děláme. Světlými okamžiky byla právě praxe na počítači i přes nutnost častého a pravidelného dojíždění, vždyť po opravě jakékoli chyby se výsledek dal zjistit až v dávce druhý den. Tato výuka nám dala dobré základy během malého počtu hodin, avšak k poutavosti měla dost daleko. Kdyby to tehdy bylo dobrovolné, určitě bych o tento zážitek přišel.

Kromě algoritmizace úloh a syntaxe jazyka jsme se samozřejmě museli seznámit i s technickou podstatou věci, možnostmi vstupních a výstupních zařízení a vnitřní prezentací dat, ale jen opravdu v nutném rozsahu, který jsme také prakticky používali. Nikdo nás nenutil znát detailně strukturu toho počítače, sčítat místo něj data v oktalovém kódu nebo mít v malíčku anatomii jeho tiskáren. V tomto směru jsme přistupovali k počítači jako uživatelé, přímá obsluha byla věcí operátorů.

Zastavím se zase u přístupu k počítačům, se kterým jsme se tehdy setkávali. Ten, kdo pracuje s počítačem, je programátor (operátory provozu tiše opomíjím). Tedy - když chcete pracovat s počítačem, musíte se naučit programovat. Důležitá věta, ještě se k ní vrátím. Uživatelské programy tehdy již nepochybně v jisté formě existovaly, ale nikdy jsme se s nimi nesetkali.

Přešlo zase pár let, vlastně právě dva, možná nejvýš tři roky. Na vysoké škole jsem ještě zažil počítač PDP 11 v jakési východní kopii. Data jsme pro něj připravovali na děrné pásky - to byl panečku rachot, když se na chodbě rozjelo patnáct děrovaček současně. Učili jsme se Fortran a začínali psát tu a tam i programy, které k něčemu byly - tedy kromě výukových cílů.

Přes ČSVTS jsme měli možnost se docela důkladně seznámit i se stolními kalkulátory Hewlet Packard, které se vzhledem, a do jisté míry i možnostmi, již podobaly tomu, co dnes nazýváme osobním počítačem. A také tyto kalkulátory přinesly další jazyk, Basic, velebený i proklínaný, kterému mělo patřit následujících několik let.

Na stolních kalkulátorech HP jsem se poprvé setkal s čistými uživatelský-

mi programy, tedy takovými, které stačilo spustit a pak již komunikovat s programem jen na základě obecné znalosti problematiky. Byly to programy většinou získané spolu s kalkulátorem nebo napsané pracovníky katedry, ale byly to opravdu programy uživatelské. Bylo možné s nimi pracovat, řešit smysluplný reálný problém a přitom nebýt ani trochu programátorem.

Vedle počítačových zážitků „oficiálních”, tedy spojených se školou, se v té době stala vcelku nenápadná, avšak naprosto převratná věc. Bylo možné se opatřit počítač domů a dělat si s ním cokoli. Sinclair ZX-81 byl první počítač, který stál méně než kurs programování (tedy prý alespoň v Británii, u nás to bylo přece jen výrazně jinak). I u nás však odstartoval průnik počítačů do domácností.

Opravdový vrchol rozšíření osmibitových počítačů přineslo ZX Spectrum. Z dnešního pohledu a ve srovnání s nynějším standardem výkonnosti jednoduchá hračka přivedla především prostřednictvím her k počítačům obrovské množství mládeže. Ale nejen hry běžely na ZX Spectru. Objevilo se i hodně užitečných aplikačních programů. Postupem času byly na tento počítač implementovány i různé překladače jazyků a dokonce systém CP/M, který ho dál přiblížil „opravdovým” počítačům.

Snad každý majitel Spectra zkusil alespoň trochu i jeho programování. Do škol a kroužků se dostaly české PMD-85, IQ 150, PP, dovozové Atari, Sharpy MZ-80 a mnohé další počítače. Nebudu rozebírat jejich možnosti a technickou úroveň, ale svůj účel většinou plnit mohly. Neštěstím byla především roztříštěnost typů (často i v rámci jedné školy) a metodiky práce s nimi. Nekompatibilita, přílišná různorodost a setrvačnost v myšlení hodně přispěly k tomu, že tyto počítače odvedly jen zlomek práce, kterou mohly zvládnout.

Během zhruba jen deseti, dvanácti let se však naprosto změnil přístup k počítačům. Od strojů určených jen pro kvalifikované profesionální obsluhy až po hračku pro děti. I ve výuce práce s počítačem nastal obdobný posun, ale s určitým, dokonce lze říci značným zpožděním. Od kurzů pro profesionály, přes výuku na vysokých školách až po zájmové kroužky pro děti kolem deseti let. Přístup lidí a jejich myšlení se však přizpůsobuje pomaleji, než odpovídá dnešní rychlosti technického vývoje.

Poznal jsem výuku počítačů, jak se říká, z obou stran katedry. Začínal jsem občas učit době, kdy základem oficiální metodiky byla výuka jazyka Basic. Práce s počítačem se prakticky ztotožňovala s programováním v tomto jazyce. Bylo hodně lidí, kteří si uvědomovali nesmyslnost takového přístupu, jenž občas vedl i ke krajně absurdním situacím.

Zažil jsem např. na vlastní kůži kurz programování v Basicu pro funkcionáře Svazarmu, kteří byli převážně v důchodovém věku nebo těsně před ním, a většinu života vojáky z povolání, především politickými pracovníky. Kdo zažil socialistickou vojnu, ví, co tím myslím. Místa ve Svazarmu byla pak pro ně pokračováním a dokončením jejich kariéry. O projevovaném zájmu a smysluplnosti či úspěšnosti takového kurzu si těžko někdo mohl dělat jakékoli iluze. Avšak přece jen jistý výsledek to mělo. Pro mě jako lektora to byl zážitek a zkušenost, na kterou nikdy nezapomenu.

Programování jako synonymum pro práci a počítačem naštěstí časem ustoupilo, ale zastavme se u vývoje způsobu jeho výuky. Metodika nestačila sledovat rychlý vývoj techniky a přesun ke stále mladším věkovým skupinám. Postupy efektivní pro výuku dospělých šlo u dětí používat jen s obtížemi, bez výrazného přizpůsobování lektorem vůbec ne.

Pokud se podíváme na učebnice programování nebo příručky, obsahují většinou základy jazyka a příklady, samozřejmě i určité obvyklé algoritmy, základní technické pojmy a vysvětlení činnosti počítače. Typickými úlohami, které najdeme, je výpočet faktoriálu, numerických řad, hledání blízkých dvojic prvočísel, statistické výpočty, operace s maticemi apod. Všechny tyto oblasti příkladů vyžadují určité znalosti matematiky, což zásadním způsobem omezuje jejich použitelnost pro děti základních škol. I pokud odpovídající znalosti matematiky již mají, nejsou pro ně takové příklady motivující a vidí v nich jen školní příklady v jiném kabátě. Podle toho k nim také přistupují.

V poslední době převládá naprosto opačný přístup, než jaký byl před deseti lety. Dá se charakterizovat přibližně takto: „Každý by se musí naučit používat počítač”. Přitom používáním počítače, či jestli chcete, prací s ním, se většinou rozumí především základní ovládání textového editoru a tabulkového procesoru, v menší úrovni seznámení s některým grafickým programem a jistá představa o databázích. Navíc se přidává čím dál více orientace na internetu a využití elektronické pošty.

To vše je užitečné, ale odsouvání znalostí programování zcela stranou a především soustavné zdůrazňování, že „normální” lidé programovat nepotřebují a nikdy potřebovat nebudou, to pokládám za nesmysl a blud, který je ještě mnohem horší a vymstí se nesrovnatelně více, než dřívější přístup typu „co uživatel počítače, to programátor”.

Pokud si chcete ověřit, s čím jsou děti školního věku o počítačích seznamovány (pokud vůbec s nimi seznamovány jsou), projděte si třeba schválenou učebnici „Práce s počítačem” [1] pro předmět „Praktické činnosti” v šestém až devátém ročníku základních škol. Z 256 stran je programovacím jazykům a programování věnována jedna strana, na níž je velmi obecně vysvětlen pojem program a programovací jazyk. Otištěná ukázka výpisu zdrojového textu v kontextu s vysvětlením působí jedině jako zastrašující prvek ukazující hrůzu toho, co programování znamená. Z textu pak budu citovat dvě věty, z nichž druhá je v originále tučně zvýrazněna: „Programy tedy vytvářejí odborníci a ostatní uživatelé počítačů si pak tyto programy kupují a používají na svých počítačích. Běžný uživatel, kterým jsi nejspíš i ty, proto nemusí mít o způsobech programování ani tušení.”

Abych této knize nekřivdil, v jiné části jsou věnovány další čtyři strany povšechnému povídání o programování v Logu, autorských systémech a o tom, že programování vyžaduje logické myšlení.

Programování není jen činnost, při které se zapisuje program v daném jazyce. Je to především schopnost formulovat problém a formulovat ho zcela přesně. Dále je to schopnost rozdělit větší problém na řadu menších, dělit tak dlouho, až částečné úlohy je možné vyřešit samostatně. Je to schopnost spojit drobné vyřešené úlohy zpět do fungujícího celku.

Programování je i umění hledat v problémech zákonitosti a závislosti a využívat jich, věnovat pozornost sebemenšímu detailu a současně neztrácet přehled o celku. Zápis programu v jazyce je až tím téměř posledním krokem, který v porovnání s předchozími obsahuje jen nepatrnou část znalostí a tvůrčího myšlení.

Návyky získané i jednoduchým programováním výrazně pomáhají řešit každodenní situace, které s počítači nemají vůbec nic společného. Tyto návyky se pak škola snaží pěstovat roztříštěně a po řadu let podstatně složitějším způsobem.

Ze znalostí, které se děti učí ve škole, rozhodující většinu nebudou nikdy ve svém praktickém životě potřebovat (samozřejmě mimo takových věcí, jako znalost čtení, psaní a základní matematiky, či přesněji řečeno aritmetiky). Přesto se je učí a učit mají, protože se tím vytváří jádro jejich budoucího všeobecného přehledu. Když se pak postupem času specializují, znalosti základní školy jsou tím, na čem staví. Není možné předem určit, co budou potřebovat a co ne, ani není možné jim vnucovat specializaci již od prvních tříd. A přestože většina nikdy neupotřebí a zapomene znalosti z biologie, chemie, fyziky, literatury a mnoha dalších předmětů, přesto se tyto předměty učí.

Programování, které v sobě soustřeďuje v daleko vyšší míře než jiné činnosti schopnost analýzy problému, formulování myšlenek a logického myšlení, je předkládáno dětem jako něco, co naštěstí nikdy nebudou potřebovat. Přitom právě programování je možné učit tak, aby samo zaujalo a bylo přijímáno s chutí. Z tohoto důvodu myslím, že by se programování mělo objevit v osnovách ve větším rozsahu

v rámci předmětů seznamujících s počítačem, a rozhodně ne jako něco, před čím škola chrání a varuje.

Ještě jeden pohled na věc je velmi důležitý, a to je stále klesající věk dětí při jejich prvních kontaktech s počítačem. Je to naprosto logické. Pokud kdysi dítě sledovalo celé dny otce kováře při práci, zcela samozřejmě bušilo jakoby kladívkem do věcí kolem sebe. Vlastnost napodobovat dospělé je pro děti velmi příznačná. Stejně tak, bude-li dítě sledovat především vaření, začne si na něj také hrát. Jestliže dnes dítě vidí rodiče doma u počítače, a při návštěvě u nich v práci vidí počítače rovněž (a kde dnes počítač nenajdete ...), chce s ním „pracovat" také.

Moje osobní zkušenost je ta, že syna ve věku batolete nepřimělo nic k pohybu tak spolehlivě a rychle, jako možnost dostat se ke klávesnici. Brzy dostal svou vlastní klávesnici, vyřazenou a vyčištěnou, později i nefunkční monitor, takže měl svůj vlastní „počítač". Že na tom monitoru nic nebylo? Přes to se dětská představivost snadno přenese. Byla to určitou dobu nejmilejší hračka. Obojí stálo dohromady asi 40 Kč. Určitě to nebyl ojedinělý případ, podobné klávesnice jsem od té doby viděl v mnoha dětských ohrádkách.

Ať chcete nebo ne, děti se dnes najdou cestu k počítači samy už v předškolním věku, dřív než zvládají čtení a psaní. Přes školky, přes kamarády, doma. Je především na dospělých, jestli budou v tomto věku formovat jejich postoj a vztah k počítačové technice na hrách, kde motorovou pilou rozřezávají zaživa mutanty za řevu a stříkání krve, nebo jestli budou mít děti k dispozici tvůrčí programy a hry přiměřené svému věku. Takových již dnes lze na trhu nalézt celou řadu.

Opět neodsuzuji šmahem všechny hry, ani ty nejtvrdší bojové. Dokonce tvrdím, že v určitých situacích citelně pomáhají uvolnit vztek a agresivitu a je určitě lepší rozřezat na počítači pár potvor motorovou pilou, než seřvat spolupracovníky nebo schválně někomu dupnout na nohu.

Jaký si může vytvořit názor na počítače dítě, které od předškolního věku zná jen hry u kamarádů, u rodičů vidí především účetnictví a texty, ve škole se po několika letech učí kategorizovat druhy aplikačních programů a je zkoušeno z částí počítače a techniky, která byla zastaralá už v době vyjití učebnice? Tím vůbec nechci jakkoli napadat autory učebnic, doba napsání učebního textu, jeho schválení a vydání a rozšíření v praxi je přibližně stejná jako doba, za kterou se počítače dnes vyřazují kvůli morální zastaralosti. I proto v této oblasti záleží mnohem víc než v jiných, „stabilizovaných" oborech na lidech a na jejich přístupu.

NEPODCEŇUJTE SCHOPNOSTI DĚTÍ! Zvládnout základy programování je pro ně lehké a přirozené, stejně jako mnoho jiných věcí. Děti dokážou nesrovnatelně víc, než si třeba jen připouštíme, ale musí k tomu mít motivaci a nesmí cítit, že jsou k čemukoli nuceny. Tou motivací nejsou známky, čokoláda za kus programu nebo dokonce nátlak a hrozba trestu, ale možnost práce na počítači jako taková, práce, která má nějaký výsledek, efekt, dá se vidět a osahat.

Má-li se udržet pozornost a chuť dětí do práce, musí každá činnost, každý pokus vést k co nejefektnějšímu výsledku ve velmi krátké době, konkrétně během patnácti, nejvýše dvaceti minut. Výsledek nesmí být odkládán na příští hodinu, na příští schůzku kroužku. Pokud se v této době výsledek nedostaví, značná část dětí ztrácí zájem. Jedna z věcí, kterou děti (zejména ty menší) nedokáží a nemůžeme ji po nich ani chtít, je pracovat („dobrovolně" a „radostně") delší dobu s vizí, že to někdy později k něčemu bude.

Budeme-li uvažovat dítě předškolního věku, nelze počítat se znalostí čtení a psaní, tím méně pak matematiky. Už od čtyř let však pro dítě není problém pracovat s myší a vzájemně přiřazovat symboly na obrazovce a klávesnici - stisknout odpovídající tlačítko. S myší se dokonce většina dětí skamarádí a získá zručnost mnohem rychleji, než dospělí na počítačových kurzech.

Karel

Před lety byl na školních osmibitových počítačích populární programovací jazyk „Karel", který spočíval ve vedení figurky na čtvercových polích. Základní povely jako vlevo, vpravo, krok vpřed, polož (značku), seber (značku) bylo možno zadávat přímo k vykonání, nebo z nich sestavovat program. Program, který vyžadoval myšlení a logiku, a přitom vystačil s několika obecně srozumitelnými povely. Současně však zápis povelů slovy (ve většině implementací) omezoval dolní věk pro Karla přibližně na druhou třídu ZŠ.

Většina dětských prográmků v Karlovi se soustředila na to, že byl na ploše vykreslen jednoduchý domeček, orámovány hranice plochy apod. V dalším postupu pak pochopili žáci používání podprogramů a rozhodovacích funkcí, později rekurze. Hledala se cesta z bludiště nebo dokonce simulovaly problémy složitější, např. známé Hanojské věže. To však už opravdu nebylo pro každého. Viděl jsem třeba i výpočty matematických operací prováděné pomocí značek v Karlovi.

Tento jazyk děti většinou zaujal alespoň na nějakou dobu, výstup byl pro ně efektní (na úrovni tehdejší techniky a zvyklostí), srozumitelný, psaní programů se některým zalíbilo, jiným ne. A to je to dobré a důležité, děti dostaly možnost přiměřenou svému věku. Přístupnou formou Karel seznamoval děti s tím, co programování je a umožňoval jim si ho osahat. Koho to chytilo, našel si cestu k dalším zájmovým kroužkům a jiným programovacím jazykům, možnost využil. Koho ne, měl celkem bezbolestnou zkušenost a vlastní názor. Namísto tohoto přístupu dnes většinou škola nejen možnost programovat nedává, ale často od něj víceméně žáky odrazuje.

Omlouvám se těm školám a především těm pedagogům a dobrovolným vedoucím kroužků, kteří se právem předchozími řádky mohou cítit uraženi, protože se ze všech sil snaží, aby u nich byla situace jiná. Vím, že leckde je lepší, někde i přesně opačná, taková místa existují. Jsou však spíše vzácností.

Baltík

Pokračovatelem jazyka Karel se stal Baltík české firmy SGP Systems, s.r.o. Tento program je mimořádně



Obr. 1. Ukázka programu napsaného v jazyku Baltík

vhodný pro děti, které baví tvořivý přístup, a to je při vhodných podmínkách většina. Baltík se ovládá myší, i ke tvorbě programů stačí sestavovat obrázkové symboly. Proto je Baltík na rozdíl od staršího Karla velmi dobře zvládnutelný už pro čtyřleté děti, i když plné využití jeho možností vyžaduje vyšší věk, řekněme 10 až 12 let. Technicky nemá program velké nároky, postačí počítač 486 s Windows 95.

Prostorem, v němž se Baltík pohybuje a kouzlí, je pracovní plocha okna - scéna. S Baltíkem můžeme pracovat v několika režimech.

Nejmenší uživatelé nemusí pracovat přímo s figurku Baltíka, mohou vytvářet obrázky na scéně tím, že z rozsáhlé banky obrázků a jejich fragmentů vybírají jednotlivá pole a přetahují je na scénu, kde z nich skládají větší celky.

Když zvládnou základní zručnost práce s programem (zejména práci myší) a pochopí symboly, mohou začít využívat figurku Baltíka přímými povely zadanými kliknutím na příslušný obrázek. Baltíka vedou na zvolené místo, kde přičaruje vybraný předmět. Opakováním tohoto postupu opět vznikají především obrázky.

Nejvyšší stupeň je programování činnosti Baltíka i s možností využití dalších jednoduchých funkcí. Zásobu předmětů lze doplňovat o svoje vlastní, střídáním obrázků a jejich časováním i vytvářet jednoduché animace, což je pro děti velmi atraktivní. Přirozená dětská soutěživost je rozvíjena i vypisováním soutěží v programování. Ukázku zápisu programu najdete na obr. 1.

Děti se nenásilnou formou učí principy programování a na zadaný problém získávají přirozenou cestou pohled, který si jinak o několik let později musí pracně osvojovat.

Práce s Baltíkem je podporována i existencí vydané učebnice [2], dostupností funkční demonstrační verze programu zdarma, bezplatných seminářů pro učitele a informatiky a konec konců i nízkou cenou programu v plné verzi pro školy. Podrobnosti včetně kurzu práce s Baltíkem můžete najít třeba na stránkách *www.sgp.cz* .

Celkově lze říci, že Baltík a jeho podstatně náročnější pokračovatel Baltazar představují výborně zvládnutý a ucelený systém výuky programování pro děti, doplněný promyšlenou podporou firmy.

Protože kousek textu z úvodní internetové stránky firmy SGP věnované Baltíkovi přesně vystihuje podstatu věci, dovolím si jej se souhlasem autora doslovně citovat:

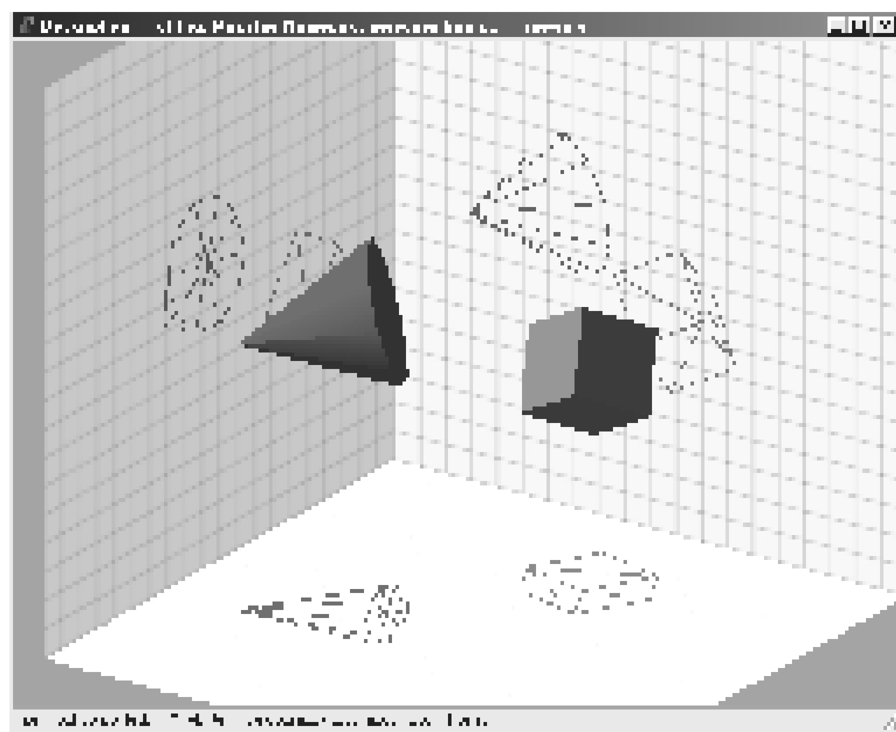
Pan Soukup vzkazuje dětem:

„Nevěřte těm dospělým, kteří vám tvrdí, že je zbytečné učit se programovat. Za pár let se totiž bude programovat téměř všechno. Už dnes si bez programování nevytvoříte ani kloudnou www stránku. Už dnes ve světě chybí 2,5 milionu programátorů. Proto si dobrý programátor ve světě už dnes vydělá 200 000 Kč měsíčně. Dobrým programátorem se ale nestanete přes noc. Pokud se chcete stát dobrým programátorem, začněte se učit včas! Začněte od základů! Začněte s Baltíkem.“

Vzkaz učitelům a rodičům:

„Nebojte se učit děti programovat. V životě se jim to bude hodit. Uvědomili jste si někdy, jaký je vlastně rozdíl mezi programátorem a uživatelem? Stejný jako mezi řidičem a cestujícím. Programátor (řidič), umí počítač ovládat, uživatel (cestující) jej umí pouze používat. Všichni víme, že každé dítě chce již od malička umět řídit. Nebraňme mu v tom, ale naopak, učme děti řídit. Až budeme v důchodu, uživatelé nás neuživí. A jenom na závěr. Nezaměstnaných uživatelů znáte jistě spousty. Ale znáte nějakého nezaměstnaného programátora? Z každého žáka jistě nebude profesionální programátor počítačů (tak jako z každého žáka autoškoly nebude profesionální řidič). To není ani naším cílem. Ale umění ovládat počítače a základy logického myšlení se budou hodit KAŽDÉMU a VŽDY.

V informační společnosti bude znalost programování velikou výhodou. Dejte svým žákům a dětem šanci!”



Obr. 2. Příklad 3D aplikace vytvořené v jazyku Petr

Petr

Dalším velmi zajímavým programem, který je schopen pokrýt ještě podstatně širší spektrum věkových kategorií i zaměření a zájmů, je programovací nástroj Petr od české firmy Gemtree software s.r.o.

Je určen pro Windows 95, 98, NT i 2000 a podporuje i tvorbu 3D grafiky, práci se zvuky, animace, operace s porty (pro nás obzvláště zajímavé), komunikace po síti, dálkové řízení programů v reálném čase apod. Velmi důležitým prvkem je zobrazování programu ve formě přehledného stromu.

Pro nejmenší děti nabízí Petr stejný způsob přístupu jako Baltík. Vytváření, přesněji řečeno sestavování obrázků na ploše, tvorbu vlastních obrázků v jednoduchém grafickém prostředí, vše je ovladatelné přetahováním grafických objektů myší.

Prostředí, které Petr nabízí, je však přece jen trochu složitější a objevují se v něm od počátku pojmy ze skutečného programování - proměnné, parametry, deklarace atd. I když je možné pro jednoduché použití obejít se bez znalosti čtení jen s významem obrázků, přece jen nesrovnatelně větší nabídka prvků dělá program pro malé děti podstatně méně přehledný.

Co je nevýhodou pro nejmenší, je neocenitelnou výhodou pro starší děti a pokročilé programátory. Petr nabízí aritmetické a logické operace, práci s textem, předměty (obrázky 32x32 bodů pokládanými na plochu), většími obrázky ve formátu BMP, pohyblivými grafickými sprajty, digitalizovaným zvukem (WAV) i syntézou hudby (MIDI).

Díky těmto vlastnostem je možné psát v tomto prostředí nejen jednoduché „placaté” hry doplněné zvuky (jako jsou známé žížaly, závody autíček, pexeso, obrázkové knížky pohádek včetně hlasového výstupu atd.), ale i 3D aplikace s viditelností (bojový simulátor tanku, zobrazení v deskriptivní geometrii apod.). Jeden z příkladů je na obr. 2.

Programy, které lze v Petrovi tvořit však nemusí nést tématicky ani na pohled sebemenší stopu po tom, v ja-

kém prostředí vznikly. Perfektním příkladem je třeba emulátor programovatelného kalkulátoru Ti-59, který zobrazuje zcela věrně klávesnici kalkulátoru. Klávesnice se ovládá myší, při stisku klávesy se ozve charakteristické lupnutí, typické pro tehdejší kalkulátory Ti. I zobrazení displeje je věrné, jen větší kvůli čitelnosti.

Plocha v okně napravo od kalkulátoru může obsahovat buď nápovědu, nebo je na ní originální termotiskárna. Opět nejde jen o symbolické naznačení, při tisku jezdí hlavička, ozývá se charakteristický zvuk a zobrazené čísla mají stejnou bodovou podobu jako originál. Dokonce i utržený papír z tiskárny má zoubkovaný okraj jako ve skutečnosti.

Ani to však není vše. Kalkulátor měl výměnné programové moduly - emulátor je má tedy také, i když jen dva. Kalkulátor uměl ukládat programy a data na magnetické štítky, i ty zde jsou ve formě souborů s programy.

Můžete jednoduše říct: Programátor si vyhrál, kde na to asi bere čas? Ale to není vůbec důležité. Tenhle program je ve svém zpracování dokonalý a vůbec nevypadá na to, že vznikl v prostředí, v němž děti staví domečky a sázejí kytičky na zahrádku.

Další, ještě konkrétnější příklady? Třeba jednoduchá databáze - adresář a telefonní seznam. Jeho vzhled i funkce je naprosto podle zvyklostí ve Windows, funkčně ani na pohled ho neodlišíte od programů, které jsou jejich příslušenstvím.

Pro naše téma je zajímavější prográmek, který vysílá na sériový port impulzy. Jejich frekvence se dá měnit, čímž se v jednoduchém připojeném přípravku mění i napětí na integračním kondenzátoru, které je pak posíleno na napětí výstupní. Výsledkem je demonstrační počítačem řízený zdroj napětí, ke kterému se připojuje malý elektromotorek.

Vrcholem ukázek u demonstrační verze programu je vývojové prostředí pro jednočipové procesory PIC16. Obsahuje editor assembleru, překladač, disassembler i programátor pro jednotku s mikroprocesorem. Velmi propracovaný je zejména systém nápověd a pomůcek při programování.

Jak už bylo řečeno, prostředí Petr není možná tak vhodné pro nejmenší a možná nemá takovou podporu zaměřenou do výuky ve školách a kroužcích, ale zato disponuje daleko širším záběrem aplikačních možností. Z toho také plyne, že v něm uživatel (postupně vyrůstající programátor) může setrvat velmi dlouho, doslova od počátků práce s myší až po technické, téměř profesionální aplikace.

Podrobnosti a aktuality naleznete na adrese *www.gemtree.cz*.

Kupte dítěti počítač ...

Dětí, které se již v raném věku dožadují vlastního počítače, je hodně. A jejich počet závratnou rychlostí roste. Většinou však toto přání vyplývá z touhy hrát si počítačové hry doma, tedy víceméně bez časového omezení hodinami vyhrazenými ve škole, v kroužku nebo při návštěvě u kamaráda.

Ačkoli to dítě nedokáže formulovat, protože ještě nezná (nejvýš tuší) relace výkonnosti počítačů a případně jejich ceny, v podstatě jeho nároky na počítač jsou velmi vysoké. Je to dáno tím, že nové hry vyžadují stále silnější hardware, zatímco běžné účetnictví a administrativu malé firmy můžete pohodlně zvládat na stroji značně zastaralém, třeba za pět tisíc z bazaru. Nároky herních počítačů míří obvykle do kategorie nejméně pětkrát dražší.

Nemusíte vyhovět, ale pokud koupíte starší a levný stroj, velmi záhy se ukáže, že na něm z nyní populárních her nechodí skoro nic - nebo nepoužitelně pomalu. Nadšení se během několika dnů změní ve zklamání a výčitky ze strany potomka a pocit nevděku na straně rodičů. V té situaci lze těžko pomoci.

Existuje však určitá alternativa ke hraní her, a to pomocí velmi silné motivace přivést děti k programování. Tato alternativa určitě není vhodná pro všechny, uplatní se zejména u dětí, které rády zkoušejí a experimentují, mají technické sklony a které bytostně potřebují si věci osahat. Alternativa i pro ty, které programy typu Karla nebo Baltíka sice na čas zaujmou, ale po určité době je omrzí, protože nevidí smysl ve vykreslování obrázků na pracovní ploše. Tou alternativou je koupit nebo sestavit dítěti jeho vlastní počítač PC, který je předem svou zastaralostí vyřazen z úvah o hraní líbivých her. Ještě lepší je nechat dítě, aby si tento počítač, samozřejmě pod stálým vedením, sestavilo samo. Není to nic složitého, vodítkem mohou být články ve starších číslech časopisu Praktická elektronika A Radio nebo některá z publikací, které se tímto tématem zabývají. Počítač používat k ovládání dětských hraček a prostřednictvím toho k jednoduché výuce programování. Šílená myšlenka? Ano, dost, ale opravdu leckdy funguje.

První námitkou tohoto postupu bývá cena. Neoprávněně. Díly na sestavení pro tento účel velmi vhodného počítače 386SX se základní pamětí 640 kB, klávesnicí, disketovou jednotkou (pevný disk jakékoli velikosti ani není nutný, ale samozřejmě se hodí), monochromatickým monitorem VGA, jednoduchou videokartou a především deskou s porty nepřevyšují v bazaru 1000 Kč - mají tedy cenu nižší, než má koloběžka nebo brusle. Velmi často se podaří takové díly nebo i většinu počítače získat zdarma, protože už ani bazary je nechtějí. Koupený celý sestavený počítač nemusí také přijít na víc, než těch 1000,- Kč.

Nízká cena má i další výhodu. Jestliže se dítěti podaří při pokusech zlikvidovat desku portů nebo přímo základní desku, není problém díl vyměnit s náklady do 50 Kč. Stejný malér na moderním počítači může hravě vyjít na stonásobek této ceny.

Další námitkou, tentokrát naprosto oprávněnou, je nutnost věnovat přípravě i následně dítěti čas. Jenže bez toho to jaksi nejde. Pokud se touto cestou vydá vedoucí kroužku ve škole, pak doba investovaná do vytvoření jednotlivých přípravků se vrátí vícenásobně a je snadno únosná, mohou pomoci i starší spolužáci s elektronickými, resp. „bastlířskými" zkušenostmi. Při přípravě jen pro jedno vlastní dítě je však časová náročnost značná.

Programové vybavení pro starý a levný počítač není celkem problém získat. Jako operační systém poslouží MS-DOS téměř libovolné verze nebo některá z jeho freewarových náhrad, pro naše potřeby zcela postačuje. Programovací jazyk bude asi nejčastěji Pascal s připravenými unitami. Opět můžeme mluvit o nejznámějším Turbo Pascalu, vhodná je i některá freewarová implementace jazyka, např. Pascal Lite Compiler for MS DOS firmy TMT.

Může být diskutabilní, zda dnes, v době nadvlády Windows a objektového programování, má smysl vracet se k DOSu a znakovému režimu obrazovky. Zda to není skok do počítačového středověku. Nepopírám, že by bylo lepší vycházet z podstatně modernějších prostředků, např. již uvedené prostředí Petr se k tomu vysloveně hodí. Jenže narazíme na tři podstatné problémy.

Prvním bude mnohonásobně vyšší cena hardware i odpovídajícího programového vybavení, druhým (třeba v případě Delphi) bude schopnost zvládnout programování obsluhy grafických vstupů a vstupů. Několik krátkých základních povelů a konstrukcí v Pascalu si dítě kolem osmi až deseti let celkem snadno zapamatuje, i když jde o slova anglická, a pak už může krůček po krůčku jít dál a seznamovat s následujícími možnostmi. Zápisy objektově orientované (třeba právě v Delphi) jsou v tomto věku podstatně hůř stravitelné. Pro vyšší věkovou skupinu, řekněme středoškoláků, je už naopak vhodnější cesta pomocí novějších prostředků. Velmi dobrým vodítkem se spoustou výborných nápadů a řešených příkladů v Delphi pak může být třeba knížka „Využití rozhraní PC pod Windows" [3].

Třetím problémem jsou zábrany, které kladou vyšší verze Windows přímému přístupu na hardware. Z hlediska běžného používání Windows a současného běhu více programů je to naprosto v pořádku, ale naši činnost to výrazně komplikuje.

Není pro nás zcela důležité, jaký konkrétní jazyk se stane základem pro obsluhu přípravků. Podstatnější je pochopit logiku jejich ovládání, dokázat obslužný program vymyslet a modifikovat podle námětů nebo prostě jen podle vlastní chuti. Práce v daném jazyce je až druhotná. Budeme zde po-

Tab. 1. Údaje vybraných vstupů a výstupů PC

Port	Typ	Adresa	Váha bitu	Označení	Číslo kontaktu CANNON 9	Číslo kontaktu CANNON 25	Název
COM 1	výstup	$3F8	x	TxD	3	2	TxD
COM 1	výstup	$3FC	1	DTR	4	20	výstup 1
COM 1	výstup	$3FC	2	RTS	7	4	výstup 2
COM 1	vstup	$3FE	16	CTS	8	5	vstup 1
COM 1	vstup	$3FE	32	DSR	6	6	vstup 2
COM 1	vstup	$3FE	64	RI	9	22	vstup 3
COM 1	GND	x	x	GND	5	7	GND
COM 2	výstup	$2F8	x	TxD	3	2	TxD
COM 2	výstup	$2FC	1	DTR	4	20	výstup 3
COM 2	výstup	$2FC	2	RTS	7	4	výstup 4
COM 2	vstup	$2FE	16	CTS	8	5	vstup 4
COM 2	vstup	$2FE	32	DSR	6	6	vstup 5
COM 2	vstup	$2FE	64	RI	9	22	vstup 6
COM 2	GND	x	x	GND	5	7	GND
LPT 1	8x výstup	$378	1 až 128	D0 až D7	x	2 až 9	výstup P
LPT 1	výstup	$37A	1	STROBE	x	1	výstup_Re1
LPT 1	výstup	$37A	2	Auto Feed	x	14	výstup_Re2
LPT 1	GND	x	x	GND	x	18 až 25	GND

stupovat v Pascalu, stejné přípravky a zejména náměty lze však stejně dobře použít i v jiném jazyce nebo prostředí.

Jedna z několika málo pomůcek, sloužících k výuce programování, vyšla pod názvem Delfín v Praktické elektronice A Radiu [4].

Využitelné porty a zdroje

Předpokládejme, že budeme mít k dispozici jeden paralelní port a dva sériové porty, což je běžné vybavení většiny PC.

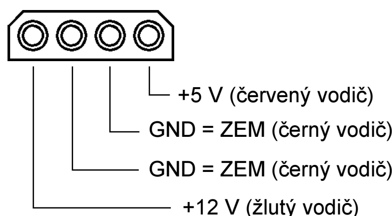
Standardní základní adresa pro sériový port COM1 je $3F8 a pro port COM2 je $2F8. K označení hexadecimálních čísel budeme v tomto textu používat znak $ před číslem v souladu s konvencí zápisu v Turbo Pascalu.

Pro nás podstatné a využívané řídicí signály (výstupy) najdeme na adresách o 4 vyšších, tedy $3FC a $2FC, vstupy na adresách o 6 vyšších ($3FE a $2FE). Čtením portů s výstupy dostaneme aktuální nastavený stav výstupů, což zjednodušuje obsluhu.

Paralelní port LPT1 má typicky adresu $378. Přímo na tuto adresu můžeme zapsat jeden výstupní byte a ten se okamžitě objeví na výstupu. Při čtení této adresy získáme naposledy zapsaný byte. Údaje pro vstupy a výstupy jsou shrnuty v tab. 1. Kompletní popis počítačových konektorů nejdeme třeba v Praktické elektronice A Radiu [5].

Při konstrukcích budeme potřebovat i napájecí napětí, většinou stačí +5 V, občas je nutných i +12 V. Napětí můžeme získat z univerzálního zdroje nebo využít přímo ta, která už máme - tedy ze zdroje pro počítač, u něhož se napojíme na konektor, určený pro velkou disketovou jednotku. Na obr. 3 jsou popsány jednotlivé kontakty tohoto konektoru. Potřebný protikus konektoru (vidlice) je běžně k dostání v prodejnách se součástkami.

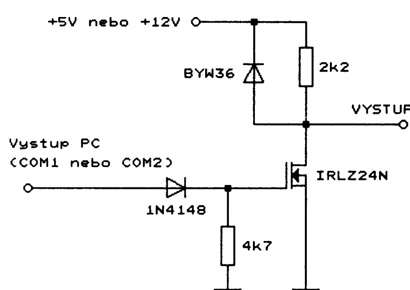
Počítačový zdroj může pro naše pokusy poskytnout proud kolem 2 A z výstupu +5 V a nejméně 0,5 A z výstupu +12 V. Při přetížení nebo zkratu zdroj vypíná svou vnitřní ochranou (čímž zhavaruje i počítač), ale v naprosté většině případů se tím sám nijak nepoškodí.



Obr. 3. Napětí na napájecí zásuvce pro FDD 5,25" v PC. Pohled na zásuvku zepředu (do dutinek)

Oddělení počítače

Výstupy z počítače je třeba pro většinu aplikací výkonově posílit. Výstupy 1 až 4, tedy výstupy brané ze sériových portů, mají při logickém stavu (úrovni) H napětí kolem +9 V, stav L vyvolá napětí kolem -9 V. Využitelná proudová zatižitelnost do 10 mA by dávala možnost napájet třeba diody LED přímo z těchto výstupů, pro řadu pokusů bychom však museli výstupy stejně posilovat. Proto je výhodné výstupy rovnou posílit s dostatečnou rezervou.

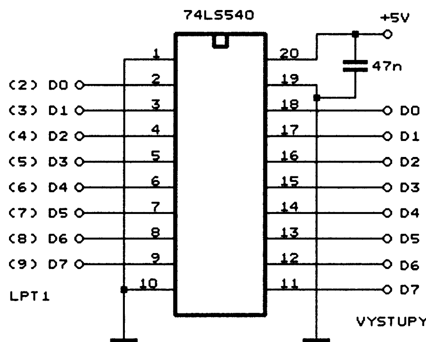


Obr. 4. Obvod pro posílení jednoho výstupu ze sériového portu

Schéma obvodu pro posílení jednoho výstupu ze sériového portu je na obr. 4, celkem toto zapojení realizujeme několikrát, čtyřikrát až šestkrát. Vstupní kapacita řídicí elektrody HEXFET tranzistoru se vybíjí přes uzemňovací rezistor o odporu 4,7 kΩ, čímž se zpomaluje uzavírání tranzistoru, pro naše účely to však není závadu. Rezistor o odporu 2,2 kΩ umožňuje při napájení ze zdroje +5 V využívat výstup i jako logický, dioda BYW36 (nebo podobná) zachytává napěťové špičky při indukční zátěži.

Použitý tranzistor IRLZ24N dovoluje s rezervou využívat výkon počítačového spínaného zdroje pro pokusy. Snese bez chladiče trvale proud až 2 A, krátkodobě přes 15 A. K jeho otevření stačí napětí menší než 5 V na řídicí elektrodě, což je velmi výhodné. Pokud by bylo potřeba větší proudové posílení, lze bez změny zapojení nasadit třeba tranzistory IRL2203N a umístit je na masivní chladiče. S nimi není problém dosáhnout trvalého proudu 30 A z jednoho výstupu, ovšem samozřejmě s úplně jiným zdrojem napájení.

Namísto tranzistorů určených pro přímé buzení z obvodů TTL lze v tomto případě použít i např. BUZ10 nebo IRF520, které se podstatně lépe shání v obchodech. Je však vhodné předem



Obr. 5. Připojení IO 74LS540

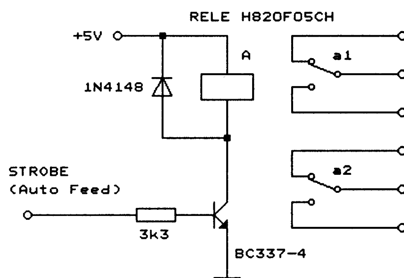
zkontrolovat, že k jejich plnému otevření stačí napětí na řídicí elektrodě +9 V nebo menší. Proudová zatižitelnost bude menší a oteplení větší.

Zátěž zapojujeme mezi výstup z posilovače a napájecí sběrnici +5 V nebo +12 V. Ke stejné sběrnici musíme propojkou připojit i kolektorový rezistor (2,2 kΩ) v posilovači. Uvedení výstupu z počítače do stavu „log. 1" (H) má za následek aktivaci zátěže.

Vstupy sériových portů není potřeba ošetřovat. Jsou stavěny na napětí nejméně ±15V, napětí větší než +3 V vyhodnocují jako H, napětí menší než asi 0,6 V jako úroveň L. I když právě uvedené údaje neodpovídají normou předepsaným úrovním na vstupech sériových portů, nesetkal jsem se v praxi s případem, kdy by to nefungovalo.

V nepřipojeném stavu je na vstupu napětí velmi blízké zemi (GND), přečteme tedy stav L. Tím se tyto vstupy liší od vstupů obvodů TTL, LS, HC nebo HCT, které nepřipojené přejdou do stavu H.

Výstupy paralelního portu je nutné ošetřit (jsou, mimochodem, mnohem náchylnější na poškození). Budeme využívat především datové výstupy D0 až D7, na které připojíme invertující budič sběrnice 74LS540. Případnou inverzi dat ošetříme programově. Je výhodné tento budič umístit do objímky, aby v případě zničení šel snadno vyměnit. Schéma připojení IO 74LS540 je na obr. 5.



Obr. 6. Připojení relé se dvěma přepínacími kontakty k výstupům STROBE a AutoFeed

Z ostatních vstupů a výstupů dostupných na paralelním portu využijeme ještě dva, a to signály STROBE a AutoFeed, na které připojíme relé se dvěma přepínacími kontakty podle obr. 6. Tato relé můžeme použít ke spínání v případě, že potřebujeme ovládané obvody galvanicky oddělit. Častěji se však uplatní při přepínání směru pohybu stejnosměrných motorků. Proudová zatižitelnost kontaktů relé by měla být alespoň 5 A.

Oddělené výstupy D0 až D7 můžeme používat samostatně, buď jako osmici reprezentující výstupní byte (třeba pro převodníky D/A nebo A/D), a nebo jako dvě čtveřice s připojenými jednoduchými odporovými převodníky D/A podle obr. 7. Tyto čtyřbitové převodníky generují napětí 0 až 1,8 V, které je operačním zesilovačem ve funkci komparátoru s hysterezí porovnáváno s pilovitým napětím. Tím se generuje signál s proměnlivou střídou pro řízení otáček motorů.

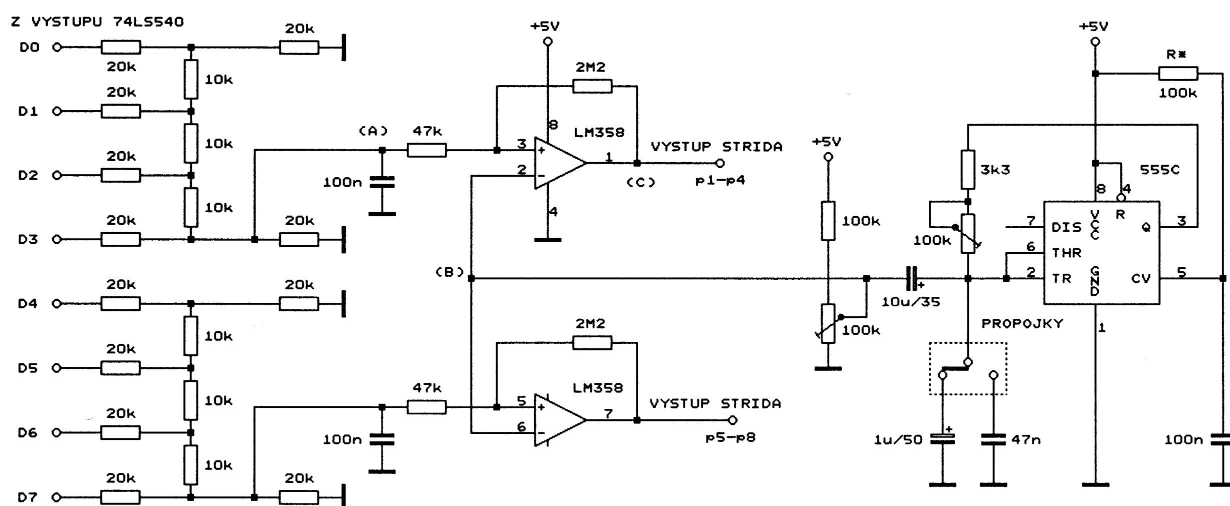
Zdrojem pilovitého napětí je časovač 555C (CMOS) zapojený tak, aby jeho výstupní signál měl střídu 1 : 1. Trimr o odporu 100 kΩ spolu s rezistorem o odporu 3,3 kΩ a se dvěma časovacími kondenzátory (1 μF, 47 nF) přepínanými propojkou umožňují nastavit kmitočet pilovitého napětí přibližně v rozsahu 10 až 200 Hz a 200 až 4000 Hz. Výstupní pilovité napětí je odebíráno přímo z časovacího kondenzátoru.

Na časovacím kondenzátoru by však byl rozkmit pilovitého napětí menší, než potřebujeme, jen asi 1,6 V a také stejnosměrná složka tohoto signálu je jiná, než jakou potřebujeme pro komparátory. Proto je mezi vývody *5* a *8* časovače 555C zapojen rezistor označený hvězdičkou, který upravuje rozhodovací úrovně časovače tak, aby se rozkmit pilovitého napětí zvětšil. Odebíraný signál je oddělen elektrolytickým kondenzátorem (10 μF) a odporový dělič s trimrem o odporu 100 kΩ mu uděluje potřebnou stejnosměrnou složku.

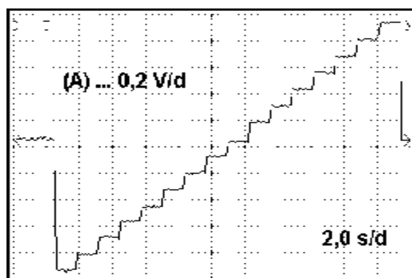
Odpor rezistoru s hvězdičkou musíme při oživování nastavit tak, aby při vstupních datech 0000 bylo na výstupu komparátoru napětí 0 V a žádné impulzy, při vstupu 0001 se objevily na výstupu komparátoru úzké impulzy, které se postupně až do stavu 1110 rozšiřují, a při vstupu 1111 bylo na výstupu komparátoru napětí kolem 5 V a opět žádné impulzy. Když je pásmo vstupních čísel, ve kterém vznikají impulzy, užší než 0001 až 1110, odpor rezistoru s hvězdičkou zmenšíme a naopak. Výchozí velikost tohoto odporu je 100 kΩ. Trimr o odporu 100 kΩ, kterým se nastavuje stejnosměrná složka pilovitého napětí, vždy dorovnáme tak, aby při vstupech 0000 a 1111 impulzy zanikly. Seřízení si vyžádá určitou dobu a je výhodou mít k dispozici osciloskop, ale není to nutné. Pro řízení motoru je samozřejmě třeba výstup z operačního zesilovače posílit obvodem z obr. 4.

Na obr. 8 je oscilogram výstupního napětí převodníku D/A (v měřicím bodě A) při jednom cyklu posloupnosti vstupních čísel od 0000 do 1111. Napětí se stupňovitě zvětšuje. Při větší rychlosti změn by se na hranách projevilo vyhlazení kondenzátorem C2 o kapacitě 100 nF. To však nevadí, protože tento převodník není určen pro rychlé změny, typicky na něm nastavíme úroveň, která je pak dlouho (několik sekund) držena.

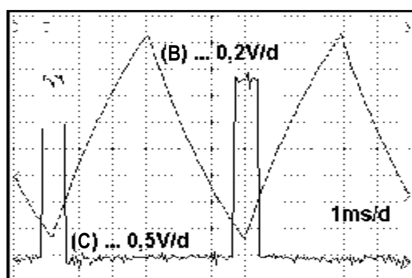
Na obr. 9 je průběh pilovitého napětí na vstupu komparátoru (v měřicím bodě B) a průběh odpovídajících impulsů na výstupu komparátoru (v měřicím bodě C) pro vstup 0010.



Obr. 7. Dva jednoduché čtyřbitové D/A převodníky s impulsními výstupy s proměnlivou střídou



Obr. 8. Průběh výstupního napětí převodníku D/A v měřicím bodě A



Obr. 9. Průběh pilovitého napětí na vstupu komparátoru (B) a průběh odpovídajících impulzů na výstupu komparátoru (C) pro vstup 0010

Uvedená zapojení je možné realizovat samostatně na univerzálních destičkách, k jednotlivým pokusům je vždy třeba jen jejich malá část.

Pro ty, kdo nechtějí používat univerzální desky, jsou všechny obvody z obr. 4 až obr. 7 (v plném počtu) soustředeny na jednu desku s plošnými spoji, která je nazvána Interface PC. Obrazec plošných spojů této desky je na obr. 10, rozmístění součástek na desce je na obr. 11. Deska navíc obsahuje dva volné univerzální posilovače, které budou použity nejčastěji pro výstupy TxD nebo operační zesilovače. Na části desky je univerzální prostor pro další doplňky. Všechny vstupy a výstupy jsou vyvedeny na svorkovnice, takže k propojování většinou stačí jen dráty na koncích odizolované a šroubováček. Pro případ, že by někdo chtěl realizovat jen část zapojení, je deska členěna na několik snadno oddělitelných částí, označených u kraje desky trojúhelníčkem.

Mechanické řešení

Desky s plošnými spoji nejsou u dalších zapojení uvedeny, protože doplňky jsou velmi jednoduché a podle možností se většinou realizují na deskách s univerzálními plošnými spoji, pokud jsou vůbec nějaké plošné spoje potřebné.

Jiné je to s mechanickým provedením přípravků a hraček, které ve většině případů bude vyžadovat více práce než elektrické zapojení. Výrazné úspory času a současně zvětšení variability získáme tím, že využijeme dílů některé dostupné dětské stavebnice. Velmi dobře se hodí LEGO, které dnes většina dětí má, zejména na různé držáky čidel. Ze starších stavebnic je neocenitelný MERKUR, především pro konstrukci různých ovládaných vozítek, samohybů a robotů.

Výhodou je spojení přípravků s hračkami, které dítě už má nebo běžně zná, především s autodráhou, elektrickým vláčkem, autíčkem (ještě lépe pásovým tankem, kde lze pohybem pásů ovládat směr) apod.

Pro náročnější (a dražší) pokusy můžeme vycházet z modelářské techniky, úpravou RC serva se dá například získat kompaktní celek stejnosměrného motorku a převodovky, výkonné pohonné modelářské motory stačí s rezervou na všechny pokusné aplikace.

Jako výborný zdroj materiálu i inspirace mohou posloužit vyřazené a nefunkční části počítačů a jejich příslušenství, především disketových jednotek, pevných disků a jehličkových tiskáren. Několik námětů můžeme najít třeba v knize „Rozeberte si PC“ [6] .

I velmi letitá literatura může být užitečná. Třeba v knize „Amatérské elektronické modely“ [7], staré již 35 let, najdeme bezpočet nápadů řešených jednoduchými zapojeními s germaniovými tranzistory, stejné nápady a pokusy však můžeme převést na řízení počítačem, zdokonalit a rozvinout.

Základní unita obsluhy

Aby se usnadnilo a pro děti zpřehlednilo psaní programů, obsahuje základní unita pod jménem „ovl.pas“ několik procedur a funkcí pro ovládání přípravků, které mají návodné české názvy. Pro zvládnutí prvních prográmků tak stačí minimum znalostí.

Unita obsahuje jako konstanty základní adresy portů com1, com2 a lpt. Před použitím kontrolujte, zda se tyto adresy shodují s nastavením vašeho počítače. Konstanty jsou použitelné i ve vlastních programech využívajících unitu ovl.pas.

Procedury se především týkají nastavování výstupních bitů, čtení vstupních bitů, jednoduchého zvukového výstupu a práce s časem. Podrobněji budou popsány vždy u prvního použití.

Ke konkrétním okruhům úloh je pak možné sestavit další unity s názvy týkajícími se přesně daného příkladu. Unitu ovl.pas a všechny uvedené očíslované prográmky je možné stáhnout z internetových stránek redakce *www.aradio.cz* .

Připravené procedury a funkce:

init - vypne všechny výstupní linky, vypne relé,

zapni1 až **zapni4** - aktivuje výstup 1 až výstup 4. Aktivace znamená spojení příslušného výstupu se zemí, tedy zapnutí zátěže zapojené mezi výstup a kladný pól napájení.

vypni1 až **vypni4** - analogicky vypnutí výstupu 1 až výstupu 4,

vstup1 až **vstup6** - funkce typu boolean - vrací stav příslušného vstupu,

zapni_p1 až **zapni_p8** - aktivuje jednotlivě výstupní bity paralelního portu p1 až p8. Aktivace znamená uvedení do stavu L na výstupu invertujícího posilovače, zátěž je typicky mezi výstupem posilovače a kladným pólem napájecího napětí 5 V,

vypni_p1 až **vypni_p8** - analogicky vypíná výstupy p1 až p8,

zapni_re1, zapni_re2 - zapne příslušné relé,

vypni_re1, vypni_re2 - vypne příslušné relé,

rychlost_1(byte) - odešle na spodní čtyři bity paralelního portu spodní čtyři bity zadaného byte. Používá se pro řízení motorů střídou impulzů přes převodník D/A. Výstupní signál se objeví na výstupu „Střída přes DA převodník p1 - p4“ na obr. 11,

rychlost_2(byte) - odešle na horní čtyři bity paralelního portu spodní čtyři bity zadaného byte. Používá se pro řízení motorů střídou impulzů přes převodník D/A. Výstupní signál se objeví na výstupu „Střída přes DA převodník p5 - p8“ na obr. 11,

vystup_p(byte) - odešle uvedený byte na datové výstupy paralelního portu. Data neguje, takže po projití invertujícím budičem jsou na výstupu v odeslané formě,

precti_p - funkce typu byte - vrací naposledy zapsanou hodnotu na paralelní port,

pockej - čeká přesně 1s,

cekej(cas) - čeká udanou dobu v desetinách sekundy,

pip - generuje tón 1000 Hz po dobu 0,1 s,

pip3 - třikrát pípne, mezery mezi zvuky jsou 0,1 s,

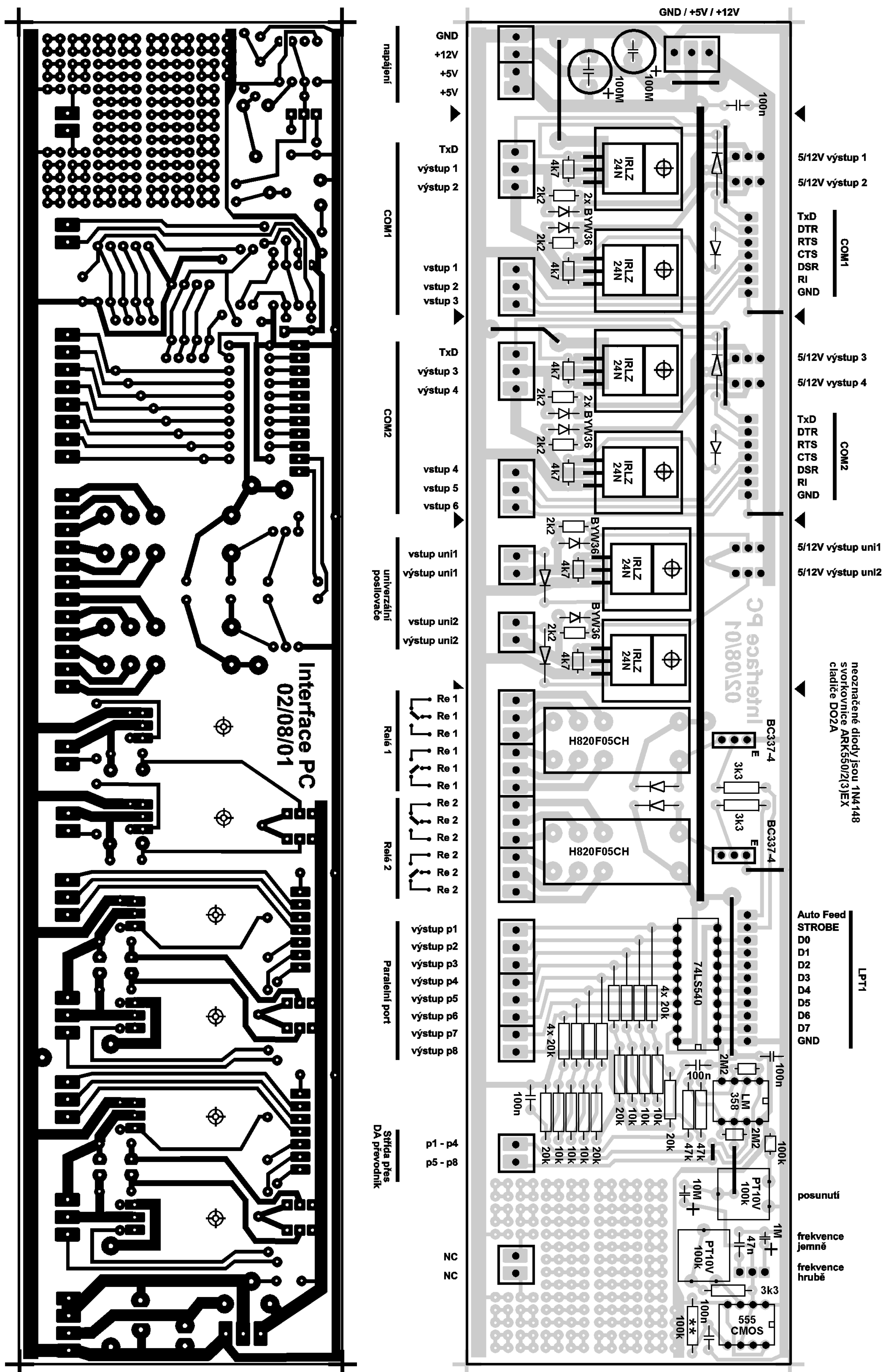
cticas - funkce typu real - vrací čas, který uplynul od začátku dne v sekundách, s přesností na 0,01 s.

Následující řada popisů přípravků a úloh si neklade za cíl být ucelenou náplní pro kroužky nebo domácí výuku, tím méně pak zpracovanou metodikou. Je to především zdroj pro vlastní inspiraci a snaží se přesvědčit, že tato cesta je použitelnou a zvládnutelnou, i když poněkud neobvyklou alternativou výuky programování.

Předpokládá se, že nejde o první setkání s programováním, že lze navazovat alespoň na minimální zkušenosti s Baltíkem nebo Petrem. Jednodušší úlohy (nebo části uvedených úloh) jsou zvládnutelné už pro děti ve věku druhé či třetí třídy, důsledné zvládnutí rozšiřujících námětů vytíží spolehlivě i středoškoláky.

Obr. 10. Obrazec plošných spojů desky Interface PC (měř.: 1 : 1) (vlevo) →

Obr. 11. Rozmístění součástek na desce Interface PC (vpravo) →


Interface PC
02/08/01
GND / +5V / +12V
GND
+12V
+5V
+5V
napájení
100M
100M
100n
TxD
výstup 1
výstup 2
COM1
vstup 1
vstup 2
vstup 3
4k7
2k2
2x BYW36
IRLZ 24N
5/12V výstup 1
5/12V výstup 2
TxD
DTR
RTS
CTS
DSR
RI
GND
TxD
výstup 3
výstup 4
COM2
vstup 4
vstup 5
vstup 6
5/12V výstup 3
5/12V vystup 4
vstup uni1
výstup uni1
univerzální posilovače
vstup uni2
výstup uni2
5/12V výstup uni1
5/12V výstup uni2
neoznačené diody jsou 1N4148
svorkovnice ARK550/2(3)EX
cladiče DO2A
Re 1
Re 2
Relé 1
Relé 2
H820F05CH
BC337-4
3k3
výstup p1
výstup p2
výstup p3
výstup p4
výstup p5
výstup p6
výstup p7
výstup p8
Paralelní port
74LS540
4x 20k
Auto Feed
STROBE
D0
D1
D2
D3
D4
D5
D6
D7
GND
LPT1
100n
2M2
LM 358
20k
10k
47k
100k
p1 - p4
p5 - p8
Střída přes DA převodník
10M
PT10V 100k
posunutí
1M
47n
frekvence jemně
frekvence hrubě
NC
NC
555 CMOS

Žárovky

Na výstupy 1 a 2 na desce Interface PC jednoduše připojíme dvě žárovky podle obr. 12. Prvním úkolem je pouze rozsvítit jednu žárovku. Nic víc. Z obecných znalostí stačí vědět, jak program začíná (begin), jak se jednotlivé povely oddělují (;) a jak se program ukončuje (end.). Dále je třeba vědět, čím se program přeloží a spustí.

Je dobré od počátku používat i hlavičku hlavního programového bloku (program „nazev") a zvyknout si nazývat programy tak, aby název vystihoval obsah. Není to sice z funkčního hlediska nutné, ale vyplatí se to. Stejně tak je dobré od samého počátku vědět, jak se do programu udělá vysvětlující poznámka - komentář ({x}).

Povely pro ovládání výstupu 1 jsou jednoduché, zapni1 a vypni1 (obdobně pro výstup 2). Žádné parametry zatím nebudeme používat. Rozsvítíme tedy žárovičku.

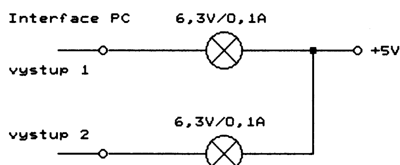
```
Program rozsvit1;
{pr_001.pas}
{rozsviti zarovku na vystupu 1}
uses ovl;
begin
 zapni1;
end.
```

Analogicky žárovku vypneme, pak třeba zapneme a vypneme druhou. První chyby jsou především formální, překlepy a nedokončená slova, chybějící středník. Je velmi zajímavé sledovat i první chyby logické, především pokusy rozsvítit již svítící žárovku nebo vypnout zhasnutou. Někteří se diví, proč se na pohled nic neděje. Neúprosná logika začíná pracovat.

Co takhle zapnout a potom vypnout obě žárovky současně? Ono to sice s připravenými procedurami nebude ve skutečnosti současně, ale bude to tak určitě vypadat.

```
Program rozsvit_obe;
{pr_002.pas}
{rozsviti obe zarovky}
uses ovl;
begin
 zapni1;
 zapni2;
end.
```

Obdobně je i vypneme. Když chceme žárovku rozblikat, potřebujeme přibrat něco, co způsobí čekání. Nejjednodušší je připravená procedura „pockej", která čeká jednu vteřinu. Zatím však nemáme ani cykly. Dokonce vystačíme i bez zcela základního pojmu proměnná.



Obr. 12. Žárovky ovládané z výstupů desky Interface PC

```
Program blikej;
{pr_003.pas}
{blikani jednou zarovkou}
uses ovl;
begin
 zapni1;
 pockej;
 vypni1;
 pockej;
 zapni1;
 pockej;
 vypni1;
end.
```

To jistě není pohodlné, zejména pro delší dobu blikání. Budeme potřebovat cyklus (pro začátek třeba s předem daným počtem průchodů) a současně i pojem proměnné jako místa v paměti počítače, kam můžeme uložit číslo (zatím stačí číslo, dokonce celé číslo). Proměnná musí mít svůj typ. Jakmile máme proměnnou, musíme ji i deklarovat. Cyklus sebou nese zase možnost vnořeného bloku. Vnořené bloky vyžadují pro přehlednost odsazování textu. To vše dohromady je značný skok, možná největší v celém postupu. Znovu tedy budeme blikat, ale delší dobu a s kratším programem.

```
Program blikej_2;
{pr_004.pas}
{blikani jednou zarovkou}
uses ovl;
var i:integer;
begin
 for i:=1 to 10 do begin
  zapni1;
  pockej;
  vypni1;
  pockej;
  end;
end.
```

Museli jsme udělat velký skok, ale zato s ním delší dobu vystačíme. Dál můžeme třeba měnit počet bliknutí v programu a zkusit blikání zpomalit (zdvojit „počkej"). Předělat blikání na druhou žárovku nebo na obě žárovky současně. A také udělat střídavé blikání obou žárovek.

```
...
zapni1;
vypni2;
pockej;
vypni1;
zapni2;
pockej;
...
```

Můžeme zkusit blikat tak, aby se žárovky střídaly, ale než se další rozsvítí, byly vždy chvíli obě zhasnuté. Není problém přidat na výstupy 3 a 4 stejným způsobem další dvě žárovky. Se čtyřmi v řadě už jde dělat přebíhání světélka jako na světelném hadu v jednom směru. Abychom vždy začínali ve stavu, který nezáleží na tom, co zbylo z předchozích pokusů, začneme používat procedur init, která vypne všechny dostupné výstupy.

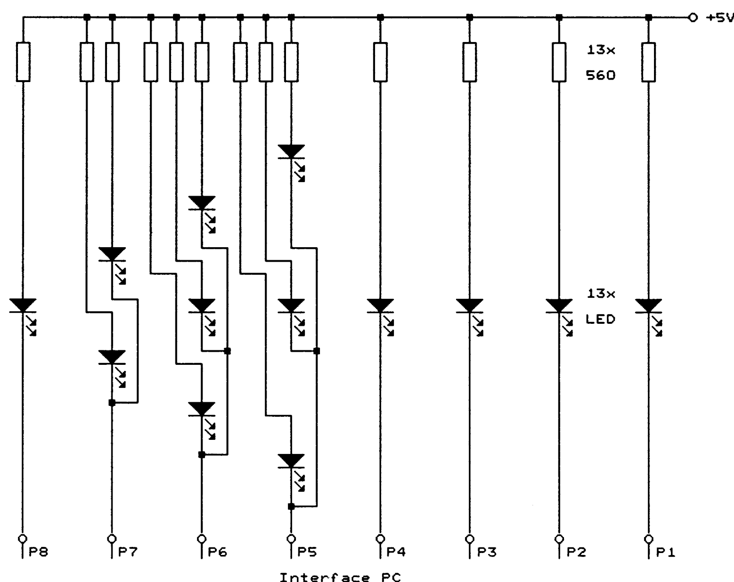
```
program had;
{pr_005.pas}
{svetelny had se ctyrmi zarovkami}
uses ovl;
var i:integer;
begin
 init;
 for i:=1 to 10 do begin
  zapni1;
  pockej;
  vypni1;zapni2;
  pockej;
  vypni2;zapni3;
  pockej;
  vypni3;zapni4;
  pockej;
  vypni4;
  pockej;
  end;
end.
```

Možnosti se dál rozšíří, když nahradíme proceduru „pockej" (s pevně nastavenou dobou na jednu vteřinu) procedurou „cekej(x)", u níž lze zadat parametrem dobu čekání v desetinách sekundy. Desetina je dobrá doba na nastavování rychlosti a současně vede k používání pro děti přijatelných velikostí čísel. Máme první jednoduchý parametr a můžeme ho využívat.

Budeme měnit rychlost blikání prostou změnou „cekej" za „pockej" na mnoha místech programu. Je to pracné, ale naprosto pochopitelné. Postupně si zavedeme proměnnou „doba" (vlastně je zatím použita jen jako konstanta), do které uložíme dobu pro čekání na jediném místě na začátku programu. Ve všech voláních „čekej" se pak můžeme na tuto proměnnou odvolat. Tím jednoduše změníme jedním zásahem do programu celou rychlost blikání, není nutné měnit program na více místech. Proceduru „init", která všechny výstupy vypíná na začátku, můžeme použít pro jistotu i na konci, abychom po sobě „uklidili".

Můžeme si vyrobit třeba světýlko, co se na konci vždy odrazí a běží zpátky, rozsvěcení celé řady z jedné strany a zase zhášení atd. Variant je bezpočet, stačí se podívat na reklamní poutače.

```
Program svetylko;
{pr_006.pas}
{svetylko beha sem a zpatky}
uses ovl;
var i,doba,pocet: integer;
begin
 init;
 doba:=4; {nastaveni rychlosti}
 pocet:=10; {pocet prebehu}
 for i:=1 to pocet do begin
  zapni1;
  cekej(doba);
  vypni1;zapni2;
```



Obr. 13. Směrová šipka ovládaná z výstupů desky Interface PC

```
    cekej(doba);
    vypni2;zapni3;
    cekej(doba);
    vypni3;zapni4;
    cekej(doba);
    vypni4;zapni3;
    cekej(doba);
    vypni3;zapni2;
    cekej(doba);
    vypni2;
    end;
  init;
  end.
```

Je to příliš pracné? Pro větší počet žárovek jistě ano. Jakmile začneme však používat procedury s více parametry, program se zkrátí, ale pro mnohé se stane už nepřehledný.

Námětem pro další práci může být model svítící směrové šipky, která bývá umístěna na stojícím autě při opravách dálnic. Rozsvěcuje se z jedné strany, okamžik svítí celá a pak zhasne. Vyrobíme ji z plastové čtvercové destičky a osadíme svítícími diodami, nejlépe žlutými. Velikost je vhodné zvolit podle nějakého dostupného náklaďáčku, který ji pak bude vozit. Protože budeme potřebovat více výstupů a postačí nám malý proud, zapojíme LED na výstupy p1 až p8 na desce Interface PC podle obr. 13.

Semafory

Semafory jsou vlastně jen obměnou a praktickou aplikací předchozích pokusů se žárovkami. Hodí se postavit skutečný malý semafor s barevnými kryty pro žárovky (jsou lépe vidět) nebo žárovky zaměnit za velké rozptylné LED (se sériovými rezistory o odporu 180 Ω). Velikost LED záleží jen na tom, pro jakou hru má sloužit (resp. na velikosti autíček).

Je zajímavější, když je možné zasahovat přímo do běhu programu. Proto připojíme mezi vstup 1 a napájecí sběrnici +5 V vypínač, který při obsluze semaforu použijeme.

V nejjednodušší verzi postačí jen přepínat barvy, samozřejmě svit žluté je podstatně kratší než červené a zelené. Semafor ale může být i „mimo provoz" a pouze žlutě blikat. Do tohoto režimu ho přepneme právě vypínačem na vstupu 1. Na obr. 14 je jeden z možných časových diagramů funkce semaforu, který odpovídá následujícímu programu.

```
Program semafor;
{pr_007.pas}
{jednoduchy semafor}
uses ovl,crt;
var doba,i:integer;
begin
 init;
 doba:=5; {nastaveni rychlosti}
 repeat
 if vstup1 then
  {blikani zlute}
  for i:=1 to 6 do begin
   zapni2;cekej(doba);
   vypni2;cekej(doba);
   end
  else begin
    {cervena/zluta/zelena/zluta}
   zapni1;cekej(doba*5);
   vypni1;zapni2;cekej(doba);
    vypni2;zapni3;cekej(doba*5);
   vypni3;zapni2;cekej(doba);
   vypni2;
   end;
 until keypressed;
 if readkey=#0 then readkey;
 init;
end.
```

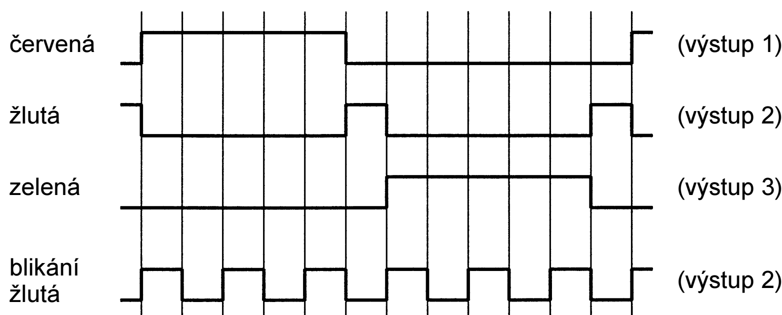
Program pracuje v cyklu dvanácti časových úseků, jejichž délka je určena hodnotou proměnné doba. Pouze po dokončení tohoto cyklu se testuje stav vypínače, jeho přepnutí se tedy neprojeví okamžitě. Používáme standardní unitu crt, která nám umožňuje odpoutat se od pevného počtu cyklů semaforu a pracovat nepřetržitě do prvního stisku tlačítka na klávesnici. Zatím jednotlivé klávesy nerozlišujeme. Na konci programu je dobré odstranit (načíst) z bufferu stisknutou klávesu, aby nebyla předána následujícímu programu.

Připojení na sériové porty výkonově posílené limituje počet použitých světel. Abychom zvládli obsluhu většího celku, použijeme výstupy p1 až p5 na paralelním portu a vyrobíme kompletní přechod s tlačítkem pro chodce na rovné silnici.

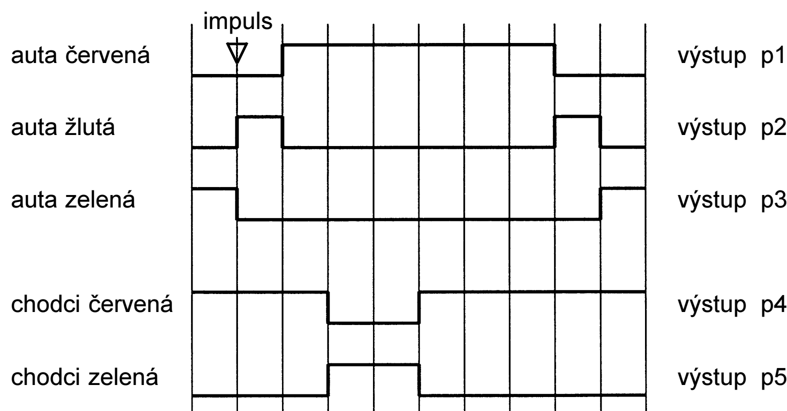
Jednotlivá světla semaforů postačí z LED o průměru 3 mm, zapojíme je jako v předchozích případech mezi příslušný výstup a napájecí sběrnici +5 V přes rezistor o odporu 470 Ω. Celkem budeme potřebovat čtyři semafory, dva pro auta a dva pro chodce. Příslušné LED (včetně rezistorů) semaforů jsou spojené paralelně, posílení integrovaným obvodem to bez problémů zvládá.

Program už vyžaduje hlubší zamyšlení nad synchronizací jednotlivých světel a případně rozkreslení do fází (stavů) na čtverečkovaný papír. V klidovém stavu (bez chodců) mají auta zelenou, chodci červenou. Stiskem tlačítka na vstupu 1 se odstartuje jeden cyklus semaforů, oproti předchozímu příkladu však musíme brát v úvahu i prodlevu na to, aby auta mohla bezpečně zabrzdit nebo opustit přechod před rozsvícením zelené pro chodce a ještě mnohem delší prodlevu mezi rozsvícením červené pro chodce a uvolněním jízdy aut. Možný časový diagram je na obr. 15.

Takto koncipované řešení už je funkční a vypadá pěkně, ale neřeší problém mnoha chodců za sebou, kteří by zcela zablokovali průjezd aut. Je ho možné vyzkoušet jako částečné řešení. Dál zavedeme proměnnou „prodleva" a konstantu „zpozdeni", pomocí nichž zajistíme, aby minimální doba zelené pro auta byla určena konstantou „zpozdeni". Pokud chodec stiskne tlačítko po delší době klidu,



Obr. 14. Jeden z možných časových diagramů funkce semaforu



Obr. 15. Příklad časových diagramů funkcí dvou semaforů (jednoho pro automobily a druhého pro chodce)

dostane volno obratem. Pokud neuplynula od předchozí zelené pro chodce doba „zpozdeni", musí chvíli počkat.

```
Program prechod;
{pr_008.pas}
{prechod pro chodce s tlacitkem}
uses ovl,crt;
const zpozdeni=50;{x0.1s}
 doba=5;{x0.1s}
var chodec:boolean;
 prodleva:integer;

begin
 init;
 chodec:=false;
 prodleva:=zpozdeni;
 repeat
  if chodec and
   (prodleva>=zpozdeni) then begin
   vypni_p3;zapni_p2;cekej(doba);
   vypni_p2;zapni_p1;cekej(doba);
   vypni_p4;zapni_p5;cekej(doba*2);
   vypni_p5;zapni_p4;cekej(doba*3);
   vypni_p1;zapni_p2;cekej(doba);
   vypni_p2;zapni_p3;
   chodec:=false;
   prodleva:=0;
   end
  else begin
   zapni_p3;zapni_p4;
   cekej(1);
   if vstup1 then chodec:=true;
   prodleva:=prodleva+1;
   if prodleva>zpozdeni then
    prodleva:=zpozdeni;
   end;
 until keypressed;
 if readkey=#0 then readkey;
 init;
end.
```

Ani to však nemusí být hotová úloha. Přechod může být mimo provoz, semafor pro chodce zhasnutý a pro auta blikající žlutá. Obdobně jako v příkladě 7 (`pr_007.pas`) můžeme doplnit vypínač na vstup 2 a ovládat blikání.

Je potom program zcela kompletní? Vzpomeňte si třeba na zvukovou signalizaci pro nevidomé na přechodech. Nasimulovat samostatně toto rychlé „klepání" při zeleném světle a pomalé při červeném třeba pomocí standardního zvukového výstupu a reproduktoru v počítači není nijak těžké, mnohem těžší je skloubit zvukovou signalizaci s časováním semaforu tak, aby vše chodilo současně jako ve skutečnosti. Můžeme zvolit dvě základní cesty - generovat zvuk elektronickým obvodem nebo čistě programové řešení.

Na obr. 16 je schéma jednoduchého generátoru s obvodem 555, který přes tranzistor budí malý reproduktorek. Krátké impulzy vytváří celkem věrně dojem klepání skutečného zařízení. Změna frekvence signálu je dosažena různými odpory v obou vstupech obvodu. Invertory z obvodu 74HC00 jsou zařazeny do vstupů kvůli tomu, aby při vypnuté signalizaci pro chodce (blikání žluté pro auta) zvukový signál utichl. Bez invertorů (a s prohozenými vstupy) funguje obvod také, ale při vypnuté signalizaci se ozve rychlé klepání, tedy povel volno pro nevidomé. Chudáci chodci!

Program prechod naprostou většinu času tráví při čekání v proceduře „cekej". Při čistě programovém řešení tedy musíme právě tuto proceduru nahradit jinou, která bude současně generovat zvukový signál. Čím pomalejší bude počítač, na kterém program poběží, tím více budou znát nepravidelnosti ve zvuku. Ale nijak výrazná tato zaškobrtnutí být nemusí.

Technicky musíme trochu předběhnout a použít prostředky, kterými se zabývá až následující kapitola.

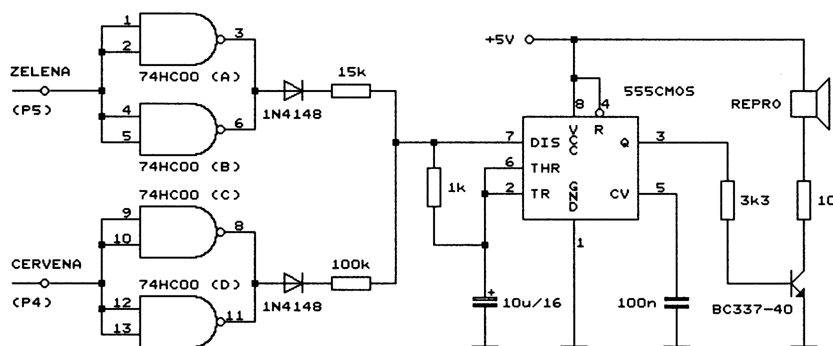
```
Program prechod2;
{pr_009.pas}
{Prechod pro chodce s tlacitkem}
{a zvukovou signalizaci}
uses ovl,crt;
const zpozdeni=80;{x0.1s}
 doba=20;        {x0.1s}
var chodec:boolean;
 prodleva,tuk:integer;

procedure cekej_z(d:integer);
{cekani s rychlym tukanim}
begin
 repeat
  delay(182);sound(400);
  delay(8);nosound;
  d:=d-2;
 until d<=0;
 end;

procedure cekej_c(d:integer);
{cekani s pomalym tukanim}
begin
 repeat
  delay(892);sound(400);
  delay(8);nosound;
  d:=d-9;
 until d<=0;
 end;

procedure prejdi;
begin
 vypni_p3;zapni_p2;cekej_c(doba);
 vypni_p2;zapni_p1;cekej_c(doba);
 vypni_p4;zapni_p5;cekej_z(doba*2);
 vypni_p5;zapni_p4;cekej_c(doba*3);
 vypni_p1;zapni_p2;cekej_c(doba);
 vypni_p2;zapni_p3;
 chodec:=false;
 prodleva:=0;
 end;

procedure klid;
begin
 tuk:=900;
 zapni_p3;zapni_p4;
 repeat
  dec(tuk);
  if vstup1 then chodec:=true;
  if tuk=8 then sound(400);
  if tuk=0 then nosound;
  delay(1);
  inc(prodleva);
  if prodleva>(zpozdeni*100)
   then prodleva:=(zpozdeni*100);
 until tuk=0;
 end;
```



Obr. 16. Generátor zvukové signalizace k semaforu pro chodce. Reproduktor má impedanci 8 Ω

```
procedure vypnuto;
 begin;
  vypni_p3;vypni_p4;
  zapni_p2;cekej(3);
  vypni_p2;cekej(12);
  tuk:=800;
  end;

begin
 init;
 tuk:=800;
 chodec:=false;
 prodleva:=zpozdeni*100;
 repeat
  if vstup2 then vypnuto else
   if chodec and
   (prodleva>=(zpozdeni*100))
    then prejdi else klid;
 until keypressed;
 if readkey=#0 then readkey;
 init;
end.
```

I toto poslední uvedené řešení však může být přepracováno a podstatně zdokonaleno. Například lze umístit na „silnici" v jisté vzdálenosti od přechodu fotočidlo, které bude počítat projíždějící auta a podle hustoty provozu bude pružně upravovat propustnost semaforu pro auta - tedy minimální dobu, kterou svítí zelená. To už se ale program začne opravdu trochu komplikovat. Kromě toho, takové vymoženosti nejsou ani na skutečných silnicích běžné.

Tento příklad se snažil ukázat, že i na tak jednoduché, běžné a samozřejmé věci, jako je semafor, lze hledat a najít řadu vztahů, které je nutné pochopit a možné zakomponovat do řízení hračky. Napsat program už není potom ani příliš náročné na použité prostředky. Jistá míra zjednodušení je možná a pro děti i naprosto nutná, ale měli bychom se ke zjednodušení uchylovat vědomě, ne z neznalosti. Čím více se budeme pokoušet přiblížit realitě, tím více podobných drobných „komplikací" budeme muset překonávat.

Zvuky

I nejstarší počítače PC jsou vybaveny malým reproduktorkem pro zvukový výstup. Signál je typicky generován vnitřním časovačem dělením vyššího základního kmitočtu. Kromě toho existují mechanizmy, jak na tento reproduktor dostat prakticky jakýkoli zvuk včetně řeči, i když v dost nevalné kvalitě.

Budeme využívat zvukový výstup především prostřednictvím běžné procedury „sound" z unity „crt". Nejjednodušším pokusem je vygenerovat tón 440 Hz (tón a1) na dobu jedné sekundy. Nesmíme zapomenout tón vypnout, jinak bude počítač pištět i po skončení programu.

```
...
sound(440);
pockej;
nosound;
...
```

Tuto možnost zvukové signalizace výborně využijeme třeba při testech nově připojovaných spínačů, tlačítek nebo různých čidel na vstupy přípravku. Program, který využívá z unity „ovl" proceduru „pip" (tón 1000 Hz po dobu 0,1 s) může vypadat třeba takto:

```
Program test_vstupu;
{pr_010.pas}
{Generuje ton pri aktivaci vstupu1}
uses ovl,crt;

begin
 init;
 repeat
  if vstup1 then pip;
 until keypressed;
 if readkey=#0 then readkey;
end.
```

Musíme brát v úvahu, že tento program by měl zaregistrovat i na velmi krátké sepnutí vstupu, proto je třeba, aby v klidu testoval vstup co nejčastěji. Z toho důvodu není v tomto případě v cestě programu žádné čekání a rychlost, s jakou vstup testuje, je dána jen možnostmi počítače. Jakmile narazí na sepnutý stav, pípne.

Pro případ, že by někdo chtěl sestavit z takto generovaných tónů jednoduchou melodii, jsou v tab. 2 potřebné kmitočty tónů v Hz pro proceduru „sound" v rozsahu tří oktáv. Je to však dost pracné a výsledek neodpovídá snaze, zaokrouhlení kmitočtů na celé Hz je příliš hrubé a výsledek „tahá za uši". Na jednoduchou fanfáru nebo pár tónů známé melodie to však stačit může.

Rychlým střídáním tónů můžeme snadno generovat různé klouzavé zvuky, kolísavé sirény a podobně. I když tyto pokusy nemají pro návaznost na další práci nijak důležitý význam, jsou u dětí bez rozdílu věku velmi oblíbené a mají jednu nespornou přednost. Dají se dělat kdykoli i bez oddělujícího přípravku na holém počítači.

Příklad pěti z mnoha možných zvuků obsahuje následující příklad. Jednotlivé zvuky se vyvolají z klávesnice tlačítky „1" až „5". Program současně ukazuje i detekci kláves a jiný způsob ukončení programu (mezerník). Program generuje výstražný signál složený z rostoucích tónů, kolísavou policejní sirénu, zvonění telefonu, tikání hodin a řadu náhodných zvuků.

```
Program zvuky;
{pr_011.pas}
{Efekty pod klavesami 1 az 5}
uses ovl,crt;
var tl:char;
 i,j:integer;

begin
 repeat
  nosound;
```

Tab. 2. Kmitočty vybraných tónů v [Hz]

Tón	c	c1	c2
c	131	262	523
cis, des	139	277	554
d	147	294	587
dis, es	156	311	622
e	165	330	659
f	175	349	698
fis, es	185	370	740
g	196	392	784
gis, as	208	415	831
a	220	440	880
his, b	233	466	932
a	247	494	988

```
  repeat until keypressed;
  tl:=readkey;
  case tl of
  '1':
   {rostouci ton}
   for j:=1 to 6 do
    for i:=50 to 600 do begin
     sound(i*2);delay(2);end;
  '2':
   {kolisajici sirena}
   for j:=1 to 6 do begin
    for i:=200 to 500 do begin
     sound(i*3);delay(1);end;
    for i:=500 downto 200 do begin
     sound(i*3);delay(1);end;
    end;
  '3':
   {zvoneni}
   for j:=1 to 3 do begin
    for i:=1 to 11 do begin
     sound(300);delay(50);
     sound(580);delay(50);
     end;
    nosound;
    delay(1400);
   end;
  '4':
   {tikani}
   for j:=1 to 16 do begin
    sound(1000);delay(3);
    nosound;delay(226);
    sound(400);delay(5);
    nosound;delay(250);
   end;
  '5':begin
   {nahodny zvuk}
   randomize;
   for j:=1 to 100 do begin
    sound(random(1000));
    delay(random(80));
    end;
   end;
  ' ':begin
    if readkey=#0 then readkey;
    exit;{ukonceni programu}
    end;
  else pip;{ostatni klavesy}
  end;
 until false;{nekonecna smycka}
end.
```

Tento způsob není jediný, kterým můžeme přímo počítačem generovat zvuk i bez k tomu určené karty. Lze využít třeba i výstupu TxD obou sériových portů. Celkově můžeme genero-

vat nejméně čtyři tóny současně - jeden časovačem a zvukovým výstupem, dva přes TxD sériových portů a nejméně jeden plně programově přes kterýkoli výstup.

Světelná závora

Pro mnoho aplikací se hodí optický snímač. Můžeme použít fotorezistory, fotodiody i fototranzistory. Uvedená zapojení byla vyzkoušena s fototranzistorem SFH309, který má slušnou citlivost, je malý (průměr 3 mm) a funguje s širokým spektrem světla.

Osvědčené zapojení s OZ ve funkci komparátoru je na obr. 17. Často potřebujeme společně dvě čidla, proto je zde rovnou použit dvojitý operační zesilovač. Nastavení citlivosti (resp. necitlivosti na úroveň okolního osvětlení) odporovým trimrem je společné. Výstupy operačních zesilovačů přivedeme přímo na sériové porty (vstupy 1 až 6 na desce Interface PC).

Fototranzistor SFH309 připojíme raději kratšími vodiči, tak asi do 50 cm, vodiče mezi OZ a počítačem (oddělovacím přípravkem) mohou být podstatně delší. Fototranzistor vždy zapustíme do neprůsvitné, nejlépe černé trubičky. Dá se snadno vytvořit třeba kouskem smršťovací bužírky. Omezí se tím pronikání světla ze strany a podstatně zlepší funkce. Stejně tak je vhodné zabránit vstupu světla do fototranzistoru zezadu.

Fotočidlo potřebuje ke spolehlivé funkci i zdroj světla. Jestliže záleží na nepatrné velikosti a spotřebě, lze použít vysoce svítivé červené LED, s kterými dosáhneme spolehlivé funkce na vzdálenost nejméně 20 až 30 cm. Se žárovkami jsou výsledky a dosažitelná vzdálenost podstatně větší, spektrum světla žárovky nesrovnatelně lépe čidlu vyhovuje. Postačují i žárovky 5 V/100 mA. Čidlo by mělo být zamířeno přímo na zdroj světla, který je nejlepší lehce zapustit do tmavé matné plošky.

Je nezbytné chránit čidlo před přímým světlem z umělého osvětlení napájeného střídavým napětím, ať už jde o žárovky nebo jakékoli zářivky. Toto světlo je modulováno síťovým kmitočtem a snadno vyvolá na výstupu komparátoru impulzy 100 Hz, u zářivek s měničem i kmitočty řádu kHz. Zdroj světla pro snímač musí mít stejnosměrné vyhlazené napájení, nejlépe přímo ze sběrnice +5 V nebo +12 V ze zdroje PC.

K vyzkoušení a seřízení světelné závory vyžijeme již uvedený program test_vstupu.

Jednoduché řízení stejnosměrného motorku

Malé stejnosměrné motorky můžeme přímo připojit mezi výkonově posílené výstupy sériových portů a kladný pól napájecího napětí 5 V nebo 12 V. Pokud používáme jako zdroj proudu spínaný zdroj PC, nesmí odběr (ani při všech motorech běžících současně) překročit povolené zatížení zdroje. Obecně však lze doporučit, aby se motorky napájely z jiného zdroje. Malé a levné motorky nemívají ani uhlíky a jejich plechové kartáčky při činnosti motoru vyvolávají takové rušení, že počítač napájený ze stejného zdroje se často „zakousne".

Výkonově posíleným výstupem můžeme nejen motor zapnout nebo vypnout, ale celkem dobře můžeme i pulzně regulovat jeho rychlost. Motor řídíme střídou vytvářených impulzů. Programem dosažitelný nejvyšší kmitočet impulzů je velmi závislý na použitém počítači, resp. jeho rychlosti. Čím vyšší kmitočet použijeme, tím je účinnost regulace vyšší, ale v nízkých otáčkách má motor malý moment a často se i samovolně třením zastaví. Naopak při nižším kmitočtu snadno dosáhneme i velmi malé rychlosti otáčení při dostatečném momentu, motor však sebou citelně škube a účinnost regulace je nižší.

Pro první pokusy s regulací otáček motoru můžeme použít následující program (`pr_012.pas`). Motor je připojen na výstup 4 na desce Interface PC. Program začíná od střídy nula (motor je vypnutý), potom v deseti krocích zvyšuje postupně otáčky motoru, mezi jednotlivými stupni pípne. Kmitočet spínání je teoreticky 100 Hz, což je pro malé motorky rozumná hodnota. Končí se střídou rovnou jedné, tedy plný během motoru.

Takto nějak by program fungoval na ideálním, velmi rychlém počítači, kdy by nebylo nutné brát v úvahu doby, které počítač potřebuje k vykonání jednotlivých instrukcí programu - třeba obsluze FOR cyklu. My však pracujeme s reálným počítačem a v případě PC 286 nebo 386SX dokonce podle současných měřítek velmi pomalým. Projeví se to na chodu motoru?

Na závěr zkoušky je přidán ještě dvousekundový plný běh motoru bez jakýchkoli impulzů, přímé sepnutí. Na rozdílu v otáčkách motoru mezi tímto posledním stupněm a stupněm předchozím jasně poznáme vliv rychlosti počítače. Čím pomalejší počítač, tím je rozdíl větší. Žádnou podstatnou změnu otáček přesto asi nezjistíme.

```
Program motor;
{pr_012.pas}
{rizeni otacek motoru}
uses ovl,crt;
var strida,i:integer;

begin
 init;pip;
 for strida:=0 to 10 do begin
  for i:=1 to 200 do begin
   zapni4;
   delay(strida);
   vypni4;
   delay(10-strida);
   end;
  pip;end;
 zapni4;
 delay(2000);
 vypni4;
 pip3;
 init;
end.
```

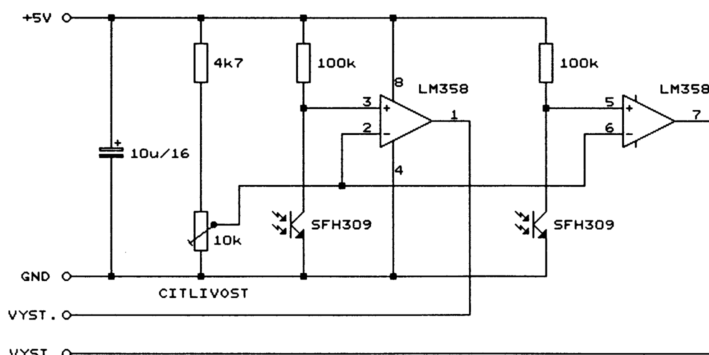
Program můžeme drobně upravit, abychom poznali, jak se motor chová při nižších kmitočtech spínání. Postačí modifikovat vnitřní FOR cyklus např. takto:

```
...
   for i:=1 to round(200/3) do
   begin
    zapni4;
    delay(strida*3);
    vypni4;
    delay((10-strida)*3);
   end;
...
```

Tím se snížil kmitočet spínání na třetinu. Postupným snižováním vyzkoušíme pro daný motor minimální kmitočet, při kterém sebou ještě příliš neškube.

Je vidět, že takto řešené impulzní řízení jednoho motoru vytíží počítač tak, že prakticky už nic jiného dělat nemůže, přestože naprostou většinu času tráví čekáním. Jakákoli další činnost musí být právě na úkor čekací doby a kompenzována tak, aby trvala vždy přibližně stejně dlouho. Jinak se na chodu motoru projeví značné nepravidelnosti. Při stejném kmitočtu je tímto způsobem možné řídit i několik motorů.

Existuje několik cest, jak nevýhodu vysokého vytížení počítače při řízení motoru zatím uvedeným „školským"



Obr. 17. Dvojice optických snímačů s fototranzistory

Tab. 3. Vztah střídy impůlzů a čísel vysílaných na sériovou linku

Střída	Start bit	D0	D1	D2	D3	D4	D5	D6	D7	Stop bit	Byte D7 až D0	Po inverzi
1/10	0	1	1	1	1	1	1	1	1	1	255	0
2/10	0	0	1	1	1	1	1	1	1	1	254	1
3/10	0	0	0	1	1	1	1	1	1	1	252	3
4/10	0	0	0	0	1	1	1	1	1	1	248	7
5/10	0	0	0	0	0	1	1	1	1	1	240	15
6/10	0	0	0	0	0	0	1	1	1	1	224	31
7/10	0	0	0	0	0	0	0	1	1	1	192	63
8/10	0	0	0	0	0	0	0	0	1	1	128	127
9/10	0	0	0	0	0	0	0	0	0	1	0	255

způsobem omezit nebo zcela odstranit. Postupně si ukážeme celkem tři.

Využití signálu TxD pro řízení

Dosud jsme museli každý vytvářený impulz spustit, počkat potřebnou dobu trvání impulzu, ukončit impulz a počkat takovou dobu, aby kmitočet opakování byl přibližně konstantní. Tedy vše obsloužit programem.

Můžeme také využít signálu TxD na sériovém portu, který jsme dosud opomíjeli. Tento výstup, který je řízen přímo z obvodu pro sériovou komunikaci, a který normálně slouží k vysílání dat, nemůžeme libovolně nastavit nebo vypnout, jako ostatní používané výstupy. Můžeme však někdy posílat z počítače taková sériová data, která vytváří požadovaný signál.

Výstup TxD sériového portu COM2 spojíme se vstupem jednoho z univerzálních posilovačů. Na výstup posilovače připojíme elektromotorek, abychom mohli výsledek bezprostředně pozorovat.

Vysílaná data se skládají ze startbitu, osmi jednotlivých bitů vysílaného byte a stopbitu. Startbit bude na výstupu vždy při vyslání dat, tvoří nejkratší dosažitelný impulz, tedy signál s nejmenší střídou. Data jsou řazena za startbitem od nejmenší váhy po největší a jsou negovaná. V sériové komunikaci se dále používá paritní bit pro zabezpečení a kontrolu přenosu dat, ten ale vypneme. Na konci dat je umístěn stopbit, z našeho hlediska nutná mezera mezi dvěma po sobě jdoucími impulzy.

Jak získáme potřebná čísla vysílaná na sériovou linku, ukazuje tab. 3. Pro výpočet v programu se lépe hodí označit si stupně střídy 0 až 8 (startbit + data = 9 bitů), vyjít z byte $FF a odrotovat ho doleva o požadovanou střídu. Bereme v úvahu samozřejmě jen jeden (spodní) byte výsledku.

```
...
b:=lo($ff shl strida);
...
```

Opakovací kmitočet impulzů nastavíme pomocí přenosové rychlosti obvodu UART. Ten má vlastní, na programu i rychlosti počítače nezávislý hodinový vstup a výsledný kmitočet našich impulzů bude 115200 (konstanta)/(dělitel x 10) (vysíláme celkem 1 + 8 + 1 bitů). Z tohoto vztahu zpětně spočítáme dělitel, který je nutné před generováním impulzů předat obvodu UART. Přesně lze protokol komunikace s tímto obvodem najít v jeho dokumentaci nebo např. v programu VIHELP, popisujícím prostředky počítačů PC a zacházení s nimi. Podrobnější vysvětlení daleko přesahuje rámec našeho tématu.

V programu je dále konstanta „doba", která určuje, jak dlouho bude každý ze stupňů ovládání motoru použit. Motor se postupně roztočí v devíti stupních rychlosti, z nichž každý trvá 2 s.

```
program motor2;
{pr_013.pas}
{rizeni motoru pomoci TxD}
uses ovl;
const frekvence=1000;{Hz}
 doba=2{s};
var strida,b:byte;
 i,delitel:word;

begin
 delitel:=round
  (115200/frekvence/10);
 for strida:=0 to 8 do begin
  pip;
  b:=lo($ff shl strida);
  {bude se vysilat rychlost}
  port[com2+3]:=128;
  port[com2+0]:=lo(delitel);
  port[com2+1]:=hi(delitel);
  {8 bitu, bez parity, 1 stopbit}
  port[com2+3]:=3;
  i:=frekvence*doba;
  repeat
   {je buffer volny?}
   if (port[com2+5] and 32)>0
    then begin
     {vysleme byte}
     port[com2]:=b;
     {pocitame impulzy}
     dec(i);
     end;
   until i=0;
  end;
 end.
```

Výhodou tohoto programu oproti předchozímu je to, že pouze odešleme příslušný byte na port COM2 a nemusíme už hlídat délku impulzu a jeho ukončení, to zařídí UART sám. Ačkoli to na první pohled z programu není zcela zřejmé, dost jsme si pomohli. Postačuje nám totiž odeslat jeden byte každou tisícinu sekundy a nemusí to být ani přesně včas, protože UART má v sobě buffer na jeden byte dat. Když tedy občas program poskytne data trochu později, na výstupních impulzech se to vůbec neprojeví.

Provedení jednoho průchodu uvedeným repeat cyklem trvá na počítači 386SX20 asi 17 µs, což je skoro šedesátkrát méně, než je potřebná perioda obsluhy motoru. Do cyklu můžeme umístit podstatně více akcí. Stačí zajistit, aby cyklus vždy proběhl za méně než 1 ms.

Abychom se zbavili i nutnosti motor vůbec nějak obsluhovat, předáme odesílání byte obsluze přerušení a budeme jen na dané místo v paměti (do dané proměnné) ukládat hodnotu vysílaného byte. Řízením motoru se budeme tedy zabývat pouze při změně rychlosti, jinak poběží vše samo. Toto řešení je podstatně náročnější na znalosti obsluhy vybavení počítače a použité prostředky jazyka, vede však k nejlépe fungujícímu programu, který zvládá řízení motoru zcela okrajově.

Využití interruptu

Základní myšlenka přerušení neboli interruptu je velmi jednoduchá. Jakmile proběhne určitá akce, vyvolá se tím signál (hardwarového) přerušení. Procesor přeruší svoji činnost, odskočí vykonat danou obsluhu přerušení a pak se vrátí a pokračuje dál.

Pro nás je podstatné, že vyprázdnění bufferu pro odesílání dat obvodu UART je jednou z akcí, která může (když se tak činnost sériového portu programově nastaví) vyvolat přerušení. V době vzniku přerušení se právě začíná odesílat další byte, máme tedy většinou dost času na to, abychom předali do bufferu další data. Tato data vezme a odešle z dané proměnné obsluha přerušení sama. Řízení motoru pod přerušením předpokládá čtyři základní kroky.

V první řadě je třeba přerušení a režim sériového portu inicializovat, tedy nastavit přenosovou rychlost a formát dat, povolit vůbec vznik přerušení, přesměrovat obsluhu přerušení na proceduru, kterou sami napíšeme a nakonec i odeslat první byte dat, kterým se odesílání rozeběhne. To vše je dost komplikovaná věc.

Za druhé musíme umět přerušení vrátit do původního stavu a odesílání

dat zastavit, protože po ukončení našeho programu nebo jeho zhavarování by bylo přerušení přesměrováno do míst, kde již žádný smysluplný program není. Důsledkem by bylo s největší pravděpodobností „zakousnutí" počítače, ale stát by se mohlo naprosto cokoli. Toto odinicializování musíme připojit k vykonání obsluhy konce programu i chybových stavů. To také není zrovna jednoduché.

Třetím krokem, který musíme ve svém programu zvládnout, je napsat vlastní obsluhu přerušení. Tu tvoří ale v porovnání s předchozím jen krátká a jednoduchá procedura - třeba tahle:

```
procedure handleinterrupt;
 interrupt;
begin
 port[$20]:=$20;
 port[comport]:=outbyte;
end;
```

Změna hodnoty byte odesílaných dat je velmi prostá - zapíšeme potřebný byte do dané proměnné, u nás se jmenuje „outbyte". Nic víc.

Většinou budeme řídit pomocí změny střídy při konstantním kmitočtu (změna odesílaných dat při stálé rychlosti komunikace). Jindy můžeme potřebovat odesílat impulzy podstatně přesnější šířky, k čemuž využijeme i řízení kmitočtu, hodí se nám ještě změna tohoto kmitočtu (frekvence) v průběhu odesílání - procedura „frekv".

Aby se zjednodušila práce se sériovým portem pod interruptem, je připravena druhá unita COMINT, která potřebné procedury obsahuje. Výpis této unity není uveden, zájemci ji najdou v souboru comint.pas. Pročtení a porozumění jejímu zápisu vyžaduje znalosti hardwaru počítače PC.

V našem programu musíme nejprve spočítat hodnotu byte, který se má odesílat a zapsat ho do proměnné outbyte (data D0 až D7 v tab. 3). K inicializaci slouží procedura „initint(cislo_portu:byte, frekvence:word) se dvěma parametry. V prvním zadáváme číslo com portu (tedy třeba 1 pro COM1), ve druhém kmitočet v Hz, jaký mají výstupní impulzy mít. Inicializace okamžitě spouští i odesílání dat.

Skutečný kmitočet vznikajících impulzů se neshoduje přesně s tím, který zadáme. Odchylka je tím větší, čím je kmitočet vyšší. Souvisí to s možností zadat pouze celočíselného dělitele do obvodu UART. Když chceme kmitočet 1000 Hz, vyjde dělitel 11520/ /1000 = 11,52, což zaokrouhlíme na 12. Skutečná kmitočet bude 960 Hz. Pokud bychom vzali dělitel 11, dostaneme 1047 Hz. Pro kmitočty kolem 100 Hz, které potřebujeme, je však už chyba menší než 1 %. Např. impulzy se střídou 1 : 1 a kmitočtem 100 Hz na COM2 vytvoříme takto:

```
...
outbyte:=240;
initint(2,100);
...
```

Není možné motor vypnout, jestliže odesílání dat běží. Vždy bude vysílán alespoň startbit (při byte $FF), který vytvoří střídu přibližně 1 : 9. Aby se motor zastavil, musíme zastavit i odesílání dat procedurou „deinitint". Tuto proceduru ale už nemusíme používat před koncem programu, sama se aktivuje jak při normálním ukončení programu, tak při chybě. Následující fragment programu by zapnul motor řízený COM2 při frekvenci 100 Hz na dostupné maximum na dobu jedné sekundy .

```
...
outbyte(0);
initint(2,100);
delay(1000);
deinitint;
...
```

Procedura „frekv(kmitočet:word)" změní kdykoli za chodu kmitočet odesílaných dat. Jako parametr použijeme požadovaný kmitočet v Hz. Můžeme třeba vyzkoušet, jak se projevuje změna kmitočtu při nejmenší dosažitelné střídě 1 : 9. Fragment programu postupně vyzkouší kmitočty 5 až 100 Hz s krokem po 5 Hz.

```
...
outbyte:=$FF;
for i:= 5 to 20 do begin
 pip;
 initint(2,i*5);
 pockej;
 deinitint;
 end;
...
```

Následující program motor3 dělá téměř totéž co program motor2, ale úplně jiným způsobem. Nechá rozběhnout motor postupně v devíti stupních, každý bude trvat 2 s. Celý cyklus se zopakuje celkem třikrát pro řídicí kmitočty 25, 100 a 400 Hz. Významný rozdíl je v tom, že obsluze motoru věnujeme pozornost velmi zřídka, v našem případě jednou za dvě sekundy při změně rychlosti, jinak program čeká nebo může dělat cokoli jiného.

```
program motor3;
{pr_014.pas}
{rizeni motoru pod interruptem}
uses comint,ovl,crt;
var strida,i:byte;
 kmitocet:word;

begin
 for i:=0 to 2 do begin
  {vypocet kmitoctu}
  kmitocet:=25*(1 shl(i*2));
  delay(2000);
  for strida:=0 to 8 do begin
   {vypocet bytu dat}
   outbyte:=lo($ff shl strida);
   pip;
   initint(2,kmitocet);
   delay(2000);
   deinitint;
  end;
 end;
end.
```

Řízení motoru převodníkem D/A

Dosud jsme se zabývali čistě programovým řízením rychlosti motoru, impulzy s požadovanou střídou vyráběl počítač. První uvedený způsob je dobře zvládnutelný a průhledný, vhodný pro pokusy, ale mnoho jiného už počítač nezvládne. Druhý je logický mezistupeň k třetímu, efektivnímu, avšak na znalosti technického vybavení PC a programové obsluhy dost náročnému způsobu. Čtvrtá metoda vyžaduje doplnění elektronického obvodu a připojení na paralelní port, zato ale je nejjednodušší na obsluhu programem a počítač vůbec nezatěžuje.

Vracíme se k původnímu záměru programování, které zvládnou i děti. I když použití procedur pro řízení motoru pod interruptem je velmi jednoduché, porozumět tomu, co se v počítači děje je náročnější a detailní pochopení unity COMINT značně obtížné.

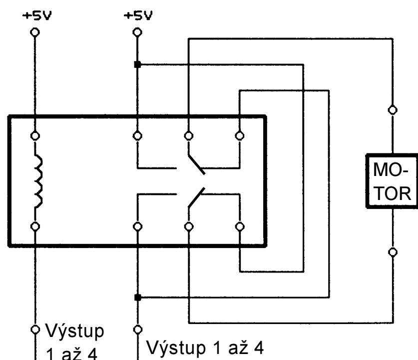
Základní myšlenka řídit motor pomocí převodníku D/A je následující. Na paralelní port odešleme byte, ten vyvolá na výstupu převodníku určité napětí. Vezmeme jiné napětí pilovitého průběhu, které si vyrobíme v obvodech a komparátorem je porovnáme. Jestliže bude napětí z převodníku menší než minimum pilovitého napětí, zůstane výstup trvale vypnutý bez impulzů. S postupně rostoucím napětím na převodníku začne komparátor pracovat a střída impulzů se bude měnit ve prospěch sepnutí motoru, až napětí na převodníku přesáhne i maximum pilovitého napětí a motor zůstane trvale sepnutý.

Použijeme-li podle tohoto popisu osmibitový převodník a velmi pečlivě obvody seřídíme, můžeme teoreticky dosáhnout stavu, kdy při zapsání bytu 0 na LPT motor stojí, v rozsahu 1 až 254 je střídou řízen jeho výkon a při 255 je motor naplno sepnut. Jemnost řízení je podstatně větší než v předchozích případech.

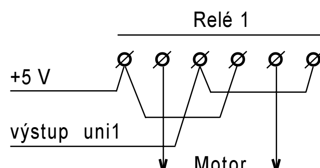
Kmitočet impulzů záleží jen na obvodech a nijak se počítače nedotýká. Stejně tak už spolu nesouvisí kmitočet a dosažitelná jemnost řízení střídy. Můžeme tedy nastavit i kmitočty vyšší, řekněme v rozsahu 1 až 10 kHz, které jsou vhodné pro motory s menší indukčností, tedy motory s vyšší proudovým odběrem a výkonem. Z hlediska programové obsluhy je tento způsob nejjednodušší ze všech. Na LPT zapíšeme odpovídající byte, to je vše.

Schéma obvodu pro řízení motoru převodníkem D/A, které je součástí přípravku Interface PC, bylo na obr. 7. Oproti popisu principu jsou použity dva čtyřbitové převodníky na jednom paralelním portu a k obsluze je připravena dvojice procedur v unitě OVL - rychlost_1(byte) a rychlost_2(byte). Zadaný byte určuje střídu impulzů na příslušném výstupu v rozsahu 0 až 15 (jsou brány v úvahu jen dolní čtyři bity z byte).

Sestavíme tedy opět podobný program, který roztočí motor postupně



Obr. 18. Zapojení relé pro přepínání směru otáčení motoru



Obr. 19. Zapojení svorkovnice relé na desce Interface PC. Relé mění směr otáčení motoru

v sekundových krocích od nuly do plných otáček. Motor je stále připojen na univerzální posilovač, vstup tohoto posilovače spojíme s výstupem p1 až p4 impulzů z převodníku A/D na desce Interface PC.

```
Program motor4;
{pr_015.pas}
{rizeni motoru D/A prevodnikem}
uses ovl;
var i:byte;

begin
 init;
 for i:=0 to 15 do begin
  rychlost_1(i);
  pip;
  pockej;
  end;
 rychlost_1(0);
end.
```

Jednoduché a přehledné obsluze motoru se nemusíme průběžně věnovat. Ale potřebujeme zdroj pilovitého napětí, komparátor a převodník.

Zatím jsme uvažovali jen regulaci rychlosti pohybu motorku bez změny směru otáčení. K doplnění této funkce bychom potřebovali rozšířit výkonové posílení na portech na plně ovládaný můstek. Pro naše účely však bude naprosto stačit doplnit relé se dvěma přepínacími kontakty zapojenými podle obr. 18. Využijeme relé na přípravku Interface PC, vývody na svorkovnici pospojujeme podle obr. 19.

Z technického hlediska to není nutné, ale měli bychom od počátku dbát na to, aby kontakty relé přepínaly ve stavu bez proudu, tedy v době, kdy motor stojí. Následující příklad pomalu roztočí motor do plných otáček, pomalu zastaví a totéž udělá i při obráceném smyslu otáček.

```
Program motor5;
{pr_016.pas}
{rizeni motoru D/A prevodnikem}
uses ovl;
var i,j:byte;

begin
 init;
 for j:=1 to 2 do begin
  for i:=0 to 15 do begin
   rychlost_1(i);
   cekej(4);
   end;
  for i:=15 downto 0 do begin
   rychlost_1(i);
   cekej(4);
   end;
  zapni_re1;
  end;
 vypni_re1;
end.
```

Doplňky k autodráze

Domácí autodráha je vděčnou hračkou pro jednoduché pokusy s připojením počítače, zejména co do možnosti vyzkoušet různé druhy čidel. Až dosud jsme v programech nepoužívali výstupu a zobrazení na monitoru PC, ten sloužil pouze k psaní programu a ovládání překladače, při vlastním běhu programu se vůbec neuplatnil. V dalších příkladech se mu už vyhýbat nebudeme, ale zůstaneme záměrně jen u nejjednoduššího použití v textovém režimu.

Smyslem uváděných příkladů je ukázat obsluhu připojených zařízení a přípravků. Jakmile se pustíme i do vytváření vzhledného projevu na obrazovce, zejména v grafickém modu, délka a pracnost programů několikanásobně vzroste. Proto rozšíření v tomto směru zůstane na chuti a zájmu uživatelů, omezíme ho na minimum. Dalším důvodem je to, že tyto části programu se značně liší nejen zápisem, ale už přístupem k řešení v různých grafických prostředích. Nicméně je rozumné po odzkoušení funkčnosti základu programů rozvíjet je dál a propracovat vzhled a komfort obsluhy tak, aby byly nejen schopné činnosti, ale i „user friendly”.

Počítadlo kol

Počítání ujetých kol patří již téměř k základní výbavě každé autodráhy. Protože většina z nich má dvě dráhy pro auta, vyrobíme rovnou snímací „most” se dvěma čidly.

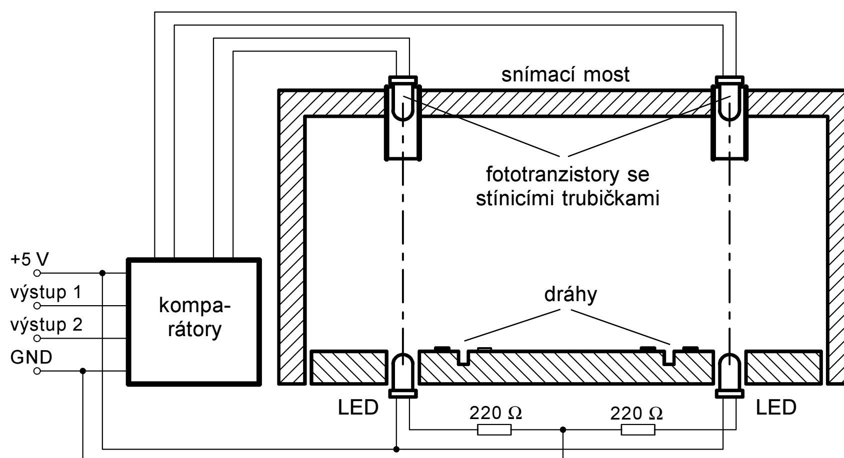
Do zvoleného dílu dráhy, pravděpodobně rovného, navrtáme dvě dírky pro zdroje světla (LED) tak, aby projíždějící auto určitě paprsek přerušilo (obr. 20). Do „mostu” nad dráhou umístíme přesně nad zdroje světla optická čidla (fototranzistory SFH309). Zásadně dáváme čidla tak, aby se dívala směrem dolů, zatímco zdroje světla dáme do dráhy, aby svítila vzhůru. To proto, že jinak by se při rozsvíceném umělém osvětlení, které bývá nahoře, síťovým kmitočtem modulované světlo dostalo do čidel a místo průjezdů autíček bychom měřili kmitočet sítě.

Při použití vhodných LED je možné udělat zdroje světla tak ploché, že se vejdou do dílu dráhy a nevyčnívají dolů. LED na místo přilepíme, výborně se osvědčila tavná pistole na roubíky plastického lepidla. Každá LED je napájená přes rezistor o odporu 220 Ω napětím 5 V.

Schéma komparátorů a zapojení fototranzistorů bylo uvedeno na obr. 17. Výstupy (VYST.) komparátorů jsou zavedeny do vstupů 4 a 5 přípravku Interface PC. Vyzkoušíme si nejprve obsluhu jednoho čidla s výpisem počtu ujetých kol, to je nejjednodušší.

```
Program prujezd;
{pr_017.pas}
{pocitani kol pro jedno auto}
uses ovl,crt;
var pocet:integer;

begin
 init;
 clrscr;
 writeln('Pocitani ujetych kol');
 writeln;
 pocet:=0;
 repeat
  if vstup4 then begin
    writeln(pocet:10);
    inc(pocet);
```



Obr. 20. Počítadlo ujetých kol - umístění LED a fototranzistorů na autodráze

```
    pip;
    while keypressed or vstup4 do;
    cekej(10);
    end;
  until keypressed;
  if readkey=#0 then readkey;
 end.
```

Čidlo je připojeno na vstup 4, auto stojí na startu kousek před ním, první průjezd (nulté kolo) je tedy bezprostředně po startu. Na rozdíl od příkladu pr_010 je zde nutné ošetřit situaci, kdy auto projíždí čidlem pomalu (nebo dokonce v místě čidla zastaví) a počítač by stihnul vyhodnotit průjezd světelným paprskem mnohokrát rychle za sebou. Proto je zde pípnutí, které reakci počítače zpomalí, a kdyby ani to nestačilo, neomezené čekání na světlený paprsek (mimochodem, během něj nelze program ukončit stiskem libovolné klávesy, jedině přes Ctrl + Break).

Ani to však nestačí, světlo může problesknout při průjezdu auta kolem kol, oknem nebo jiným otvorem a započítá se zase víc kol, než má být. Vyjdeme tedy z toho, že objetí kola trvá určitě víc než jednu sekundu, a přidáme další čekání. Teoreticky by ještě i nyní mohlo dojít k chybě při opravdu velmi pomalém pohybu, ale pokud nebudete švindlovat a snažit se chybu vyvolat, chyba už v praxi nenastane.

Program, který by počítal průjezdy pro obě auta na dráze, musíme navrhnout výrazně odlišně. V předchozím příkladě trávil počítač většinu času neustále opakovaným testováním průjezdu čidlem, počátek průjezdu tedy zachytil velmi přesně. Jakmile však zachytil počátek průjezdu, nebyl jistou dobu vůbec schopen na další podnět reagovat. Jakou dobu? Určitě čas pípnutí (0,1 s) plus dobu, po kterou průjezd auta kolem čidla překročil dobu 0,1 s plus další čekací dobu 1 s. Nejméně tedy 1,1 s po průjezdu prvního auta kolem čidla by mělo druhé auto na to, aby naprosto nepozorovaně proklouzlo kolem svého čidla.

Je třeba udělat vyhodnocení průjezdu auta tak, aby ho program jen rychle zaznamenal a okamžitě se mohl věnovat i druhému čidlu. Tentokrát také nemůžeme vypisovat počet průjezdů pod sebe a případně nechat obrazovku rolovat, záznam by byl velmi nepřehledný. Jakmile kterékoli auto projede kolem čidla, změní se počet průjezdů, vypíše se na obrazovku na stále stejné místo a současně se nastaví počítadlo testovacích cyklů „doba", po které bude jakákoli změna na daném čidle ignorována (program ji ani netestuje).

```
 Program prujezd2;
 {pr_018.pas}
 {pocitani kol pro dve auta}
 uses ovl,crt;
 const doba=2000; {ms}
 var pocet1,pocet2,doba1,
  doba2:integer;
```

```
begin
 init;
 clrscr;
 writeln
  ('Pocitani kol pro dve auta');
 writeln;
 pocet1:=0;pocet2:=0;
 repeat
  if vstup4 and (doba1=0) then
  begin
   inc(pocet1);
   doba1:=doba;
   gotoxy(3,3);
   writeln(pocet1:10,pocet2:10);
   end;
  if vstup5 and (doba2=0) then
  begin
   inc(pocet2);
   doba2:=doba;
   gotoxy(3,3);
   writeln(pocet1:10,pocet2:10);
   end;
  delay(1);
  dec(doba1);
  if doba1<0 then doba1:=0;
  dec(doba2);
  if doba2<0 then doba2:=0;
 until keypressed;
 if readkey=#0 then readkey;
end.
```

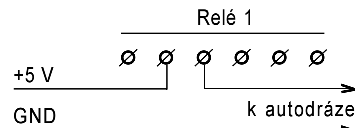
Pokud by auto zastavilo přímo pod čidlem, bude mu každé 2 s připočítáno jedno kolo. Tato vlastnost ošetřena není, může být předmětem dalšího vylepšení.

Bylo by jistě zajímavé vědět, jak rychle testování čidel probíhá a zda přece jen nemůže auto nějak proklouznout. Na počítači 386SX20 po odstranění příkazu „delay(1)" a úpravě hlavní smyčky programu na cyklus s pevným počtem průchodů trvalo 60000 průchodů asi 2,5 s - tedy průjezd auta byl testován zhruba 24000x za sekundu.

Po vrácení „delay(1)" (čekání po dobu 1 ms) trvalo totéž předpokládaných 63 s, během kterých projelo auto 16x kolem čidla. Je vidět, že i v tomto případě tráví program 95 % času v proceduře delay, ale kromě toho zvládá přibližně 1000x za sekundu otestovat čidla.

Jak daleko vlastně ujede auto po dráze za tu tisícinu sekundy? Rychlé závodní modely stavěné specialisty jezdí rychlostí i kolem 50 km/h (ty na domácích dráhách mnohem méně), což je přibližně 14 m/s. Za tisícinu sekundy ujede 1,4 cm. Je-li auto průměrně dlouhé 10 cm, je ho náš počítač schopen v této rychlosti zaregistrovat přibližně se sedminásobnou rezervou. Skutečná rychlost dětských autíček na domácích drahách je maximálně třetinová, rezerva možností našeho počítače 386SX20 vyjde asi dvacetinásobná.

Program můžeme dále rozšířit třeba tak, že počítač sám odpočítá start, začne závod připojením napájecího napětí přes relé, registruje ujetá kola a vypisuje počet kol do konce. Jakmile



Obr. 21. Zapojení svorkovnice relé na desce Interface PC. Relé spíná proud do jednoho jízdního pruhu autodráhy

jedno z aut ujede předvolený počet kol, odpojí napájení a vyhodnotí vítěze.

Napájecí napětí přivedeme na dráhu přes spínací kontakty relé 1 podle obr. 21, čidla jsou stejně jako v předchozím případě na vstupech 4 a 5. Auta stojí za čidly (než poprvé přeruší paprsek, musí objet celé kolo), každý průjezd cílem se tentokrát počítá jako kolo. Program vyžaduje zadat počet kol a očekává číslo - pokud dostane něco jiného, skončí s chybou. Zadání záporného čísla vyhodnotí jako „neplatný počet kol". Je uvažována i možnost, že by obě auta dojela do cíle současně, i když je to krajně nepravděpodobné. Uměle ji vyvoláme tak, že spustíme závod na 0 kol.

```
Program zavod;
{pr_019.pas}
{zavod na dany pocet kol}
uses ovl,crt;
const doba=2000; {ms prodleva}
var pocet1,pocet2,doba1,doba2,
 kola:integer;
procedure startpip;
 begin
  sound(1000);delay(500);
  nosound;delay(500);
 end;
procedure tus;
 begin
  sound(440);cekej(2);
  sound(494);cekej(2);
  sound(523);cekej(2);
  sound(659);cekej(8);
  nosound;
 end;

begin
 init;
 clrscr;
 write(' Zavod na dany pocet kol');
 writeln(' pro dve auta');
 tus;
 writeln;
 write(' Jaky bude pocet kol?  ');
 readln(kola);
 writeln;
 if kola<0 then begin
  writeln(' Neplatny pocet kol');
  cekej(20);
  exit;
  end;
 write(' Pripravit ke startu !');
 gotoxy(24,5);
 writeln('- 3 -');
 startpip;
 gotoxy(24,5);
 writeln('- 2 -');
 startpip;
 gotoxy(24,5);
 writeln('- 1 -');
 startpip;
```

```
  gotoxy(24,5);
  writeln('START');
  startpip;
  zapni_re1;
  writeln;
  writeln(' Draha 1    Draha 2');
  writeln;
  pocet1:=kola;pocet2:=kola;
  repeat
   if vstup4 and (doba1=0) then
   begin
    dec(pocet1);
    doba1:=doba;
    gotoxy(1,9);
    writeln(pocet1:8,pocet2:11);
    end;
   if vstup5 and (doba2=0) then
   begin
    dec(pocet2);
    doba2:=doba;
    gotoxy(1,9);
    writeln(pocet1:8,pocet2:11);
    end;
   delay(1);
   dec(doba1);
   if doba1<0 then doba1:=0;
   dec(doba2);
   if doba2<0 then doba2:=0;
  until (pocet1=0) or (pocet2=0);
  vypni_re1;
  writeln;
  if (pocet1=0) and (pocet2=0) then
   writeln(' Nerozhodne !!')
   else begin
    write(' Vitezi vuz na draze ');
    if pocet1=0 then
     writeln('jedna')
     else
     writeln('dva');
   end;
  tus;
  repeat until keypressed;
  if readkey=#0 then readkey;
 end.
```

Pokud bychom potřebovali odbourat z programu cyklické hlídání čidel a věnovat se něčemu jinému, můžeme opět případně přesunout zaregistrování průjezdu auta do obsluhy interruptu, který je aktivován změnou stavu na vstupech. Procedury pro toto řešení však připraveny nejsou a jejich vytvoření může být námětem pro další, náročnější práci.

Měření času

Pro měření času se výborně hodí možnost číst systémový čas počítače. V unitě „OVL" je připravena reálná funkce „cticas", která vrací čas v sekundách (od začátku dne) s přesností na setiny sekundy. S tímto vyjádřením času se velmi jednoduše počítá.

V následujícím příkladu je ukázka použití této funkce. Program vypíše čas počátku testu v sekundách, počká přesně jednu sekundu a vypíše čas konce testu. Teoreticky by se měly oba časy lišit právě o tu jednu sekundu, ale my asi zjistíme rozdíl poněkud větší, typicky o setinu sekundy. Dál program vyčíslí čas konce testu v obvyklém tvaru hod:min:sec. Vzhledem k určení programu není ošetřen případ, kdy v průběhu testu projde čas půlnocí.

```
Program test_casu;
{pr_020.pas}
{prace s casem}
uses ovl,crt;
var cas1,cas2:real;
 hod,min,sec:word;

begin
 clrscr;
 writeln('Pouziti funkce cticas');
 writeln;
 cas1:=cticas;
 writeln('Zacatek',cas1:12:2);
 pockej;
 cas2:=cticas;
 writeln('Konec  ',cas2:12:2);
 writeln('Rozdil ',(cas2-
  cas1):12:2,' [s]');
 writeln;
 write('Je prave   ');
 hod:=trunc(cas2/3600);
 min:=trunc((trunc(cas2) mod
  3600)/60);
 sec:=trunc(trunc(cas2) mod 60);
 write(hod,':',min,':');
 if sec<10 then write('0');
 writeln(sec);
 repeat until keypressed;
 if readkey=#0 then readkey;
end.
```

Další příklad velmi jednoduše měří čas potřebný k projetí jednoho kola na autodráze. Je vyhodnocováno čidlo na první dráze (vstup 4). Při prvním průjezdu po startu programu začne měření, po každém následujícím průjezdu se vypíše čas kola v sekundách.

```
Program cas_kola;
{pr_021.pas}
{mereni casu na jedno kolo}
uses ovl,crt;
const doba=1000; {ms prodleva}
var cas1,cas2:real;

begin
 init;
 clrscr;
 write(' Mereni casu na jedno');
 writeln(' kolo');
 cas2:=0;
 repeat
  repeat
   if keypressed then begin
    if readkey=#0 then readkey;
    exit;
    end;
  until vstup4;
  cas1:=cticas;
  pip;
  if cas2>0 then begin
   gotoxy(1,3);
   write(' Cas kola :');
   writeln((cas1-cas2):8:2,' s');
   end;
  cas2:=cas1;
  delay(doba);
 until keypressed;
 if readkey=#0 then readkey;
end.
```



Autíčko na dráze s bočním optickým čidlem (fototranzistorem)

Poněkud dokonalejší a prakticky použitelný je následující program pro vyhodnocení závodu jednoho auta na čas. Dráhu je možné sestavit s jedním nebo lichým počtem křížení tak, aby obě stopy byly využívány jedním autem. Počet kol závodu je určen pro zjednodušení konstantou. Měření začíná prvním průjezdem kolem čidla, po dojetí každého kola počítač pípne a vypíše čas kola. Průběžné časy se ukládají do pole a vyhodnocují až nakonec. Po dokončení daného počtu kol se vyhodnotí celkový čas, průměrný čas na kolo a čas nejrychlejšího kola. Zapojení čidla je stále stejné, vstup 4.

```
Program zavod_na_cas;
{pr_022.pas}
{zavod auta na cas 5-ti kol}
uses ovl,crt;
const doba=1000; {ms prodleva}
 kola=5; {pocet kol zavodu}
var i,kolo:integer;
 mincas,pomcas:real;
 cas:array[0..kola] of real;

begin
 init;
 clrscr;
 write(' Zavod jednoho auta na ');
 writeln('cas - pocet kol ',kola);
 writeln;
 kolo:=0;
 repeat
  repeat until vstup4;
  cas[kolo]:=cticas;
  pip;
  if kolo>0 then begin
   write(' Cas kola ',kolo:3,':');
   write((cas[kolo]-
    cas[kolo-1]):18:2);
   writeln(' s');
   end;
  inc(kolo);
  delay(doba);
 until kolo>kola;
 writeln;
 write(' Cas celkem :       ');
 pomcas:=cas[5]-cas[0];
 writeln(pomcas:12:2,' s');
 write(' Prumer na kolo:    ');
 writeln(pomcas/kola:12:2,' s');
 write(' Nejrychlejsi kolo :');
 mincas:=10E6;
 for i:=1 to 5 do
  if (cas[i]-cas[i-1])<mincas then
   mincas:=cas[i]-cas[i-1];
```

```
  writeln(mincas:12:2,' s');
  repeat until keypressed;
  if readkey=#0 then readkey;
end.
```

Samozřejmě je možné kombinovat, vytvořit program pro závod dvou aut s vyhodnocením vítěze a jeho času, průběžně vypisovat časy na jedno kolo obou soupeřů, vyčíslovat časovou ztrátu na vedoucího jezdce atd.

Měření rychlosti

Úloha měření rychlosti je velmi podobná měření času na jedno kolo, jen potřebujeme znát délku dráhy, kterou auto ujede. Ke změření délky okruhu se výborně hodí krejčovský metr, který vedeme drážkou v dráze.

Při optimální jízdě je nutné řídit rychlost autíčka, zpomalovat do zatáček, křížení nebo zúžení dráhy. Jaká je maximální rychlost auta na rovném úseku rychlost a jakou se projíždí zatáčka?

Optické sondy připojené ke vstupům 4 a 5 umístíme na požadovaný úsek dráhy do přesně odměřené vzdálenosti. Čím bude měřený úsek delší, tím se podaří rychlost změřit přesněji. Naopak čím bude kratší, tím menší budou změny rychlosti během průjezdu. V praxi asi bude vzdálenost přibližně rovna jednomu až dvěma dílům dráhy - tedy něco mezi 20 a 30 cm. Jednoduchý program může vypadat třeba takto:

```
Program rychlost1;
{pr_023.pas}
{mereni rychlosti auta}
uses ovl,crt;
const delka=300; {mereny usek v mm}
var start,cil,rychlost:real;

begin
 init;
 clrscr;
 repeat until vstup4;
 start:=cticas;
 repeat until vstup5;
 cil:=cticas;
 rychlost:=delka/(cil-start)/1000;
 writeln(rychlost:10:1,' m/s');
 pip;
 repeat until keypressed;
 if readkey=#0 then readkey;
end.
```

Tento program je přehledný, jednoduchý a zcela funkční, můžeme ho vyzkoušet, ale rozumně použitelné výsledky nedává. Čas zde měříme čtením systémového času procedurou cticas s přesností na 0,01s. Přitom auto při rychlosti pouhé 3 m/s, která je dosažitelná na prakticky každé dráze, projede úsek 30 cm za pouhých 0,10 s. Měření je nejen velmi nepřesné, ale má i příliš malé rozlišení.

Je třeba buď prodloužit měřený úsek nebo zpřesnit měření času. Potřebovali bychom, aby auto zůstalo v měřeném úseku asi 0,5 až 1 s, pak by chyby měření dostatečně klesly. To asi nepůjde, dostali bychom vzdálenost srovnatelnou s délkou celého okruhu. Musíme dosáhnout lepšího rozlišení při měření času.

Kdybychom měřili čas počtem průchodů nějakou jednoduchou programovou smyčkou, určitě bychom dostali nesrovnatelně lepší rozlišení. Jenže na každém počítači by taková smyčka běžela jinak rychle a se skutečným měřením času by to mnoho společného nemělo. Ledaže by se podařilo změřit počet průchodů smyčkou za přesně známý čas a potom stejnou smyčkou změřit průjezd auta. Pak bychom mohli jednoduchou trojčlenkou vypočítat skutečný čas a následně i rychlost.

Kde ale vzít ten přesný a známý časový interval na řízení kalibrační smyčky? Kde jinde než z obvodu UART, který má, jak už bylo jednou řečeno, vlastní a na všech počítačích stejný zdroj hodinového kmitočtu. Pomocí něj si potřebný interval vyrobíme.

Budeme pracovat s COM2. Nejprve si propojíme výstup TxD z COM2 na vstup 6 přípravku Interface PC. Tím máme obsazeny vstupy 4 a 5 na čidla a vstup 6 na kalibrační impulzy.

Chceme vygenerovat kalibrační impulz dlouhý přesně 1 s. Na port budeme vysílat byte 0, který při odeslání vytvoří impulz v délce 9 bitů (startbit + + 8 bitů dat) - za jednu sekundu musíme tedy vyslat přesně 9 bitů. Ze vztahu f = 115200/dělitel spočítáme dělitel - vyjde přesně 12800. Tento dělitel zadáme do UARTu.

Odešleme na sériový port byte 0. Musíme počkat, až se na vstupu 6 objeví náběžná hrana impulzu, pak smyčkou rychle počítáme průchody do proměnné pocet1, dokud impulz neskončí (tj. přesně za jednu sekundu).

```
...
repeat until vstup6;
repeat inc(pocet1) until
 not(vstup6);
...
```

Tím je kalibrace hotova. Obdobně počkáme na vjezd auta do měřeného úseku a počítáme stejnou smyčkou čas průjezdu do proměnné pocet2. Jakmile to máme, je doba průjezdu v sekundách jednoduše rovna podílu pocet2/pocet1. Počítač 386SX20 proběhne kalibrační smyčkou přibližně 70000x za sekundu, rozlišení je tedy opravdu dobré. Proměnné pocet1 a pocet2 musí pojmout větší číslo, než nám umožňují typy integer nebo word, longint však s rezervou postačuje.

Pokud budeme dělat pokusy, zjistíme, že počet průchodů kalibrační smyčkou je i na stejném PC pokaždé trochu jiný, ačkoli by teoreticky měl být stále přesně stejný. Je to proto, že v průběhu kalibrace je náš program občas odstaven a vykonává se obsluha různých přerušení souvisejících se systémem PC. Pokud bychom tomu chtěli zabránit, musíme na dobu kalibrace (a stejně pak i dobu měření) přerušení zakázat. Příslušné procedury jsou k dispozici v unitě comint (DI - zákaz přerušení, EI - povolení přerušení), kterou musíme přidat do deklarací. Vypadalo by to asi takhle:

```
...
DI;
repeat until vstup6;
repeat inc(pocet1) until
 not(vstup6);
EI;
...
```

Rozptyl naměřených hodnot způsobený interrupty je však tak malý, že ho můžeme klidně zanedbat. Následující program se zkalibruje podle rychlosti počítače, pak vypíše „START" a měří cyklicky rychlost průjezdu úsekem 200 mm dlouhým až do stisku libovolné klávesy. Program se však ukončí až následujícím průjezdem auta po stisku klávesy, protože ukončení je testováno pouze po dokončení měření.

```
Program rychlost2;
{pr_024.pas}
{presnejsi mereni rychlosti auta}
uses ovl,crt;
const delka=200; {mereny usek v mm}
var rychlost:real;
 pocet1,pocet2:longint;

begin
 init;
 clrscr;
 writeln(' Mereni rychlosti');
 writeln;
 writeln(' Probiha kalibrace');
 pocet1:=0;pocet2:=0;
 port[com2+3]:=128; {rychlost}
 port[com2+0]:=lo(12800);
 port[com2+1]:=hi(12800);
 port[com2+3]:=3; {rezim 8,n,1}
 port[com2]:=0;
 {kalibrace podle rychlosti PC}
 repeat until vstup6;
 repeat inc(pocet1) until
  not(vstup6);
 gotoxy(1,3);
 writeln(' START            ');
 {mereni prujezdu}
 repeat
  pocet2:=0;
  repeat until vstup4;
  repeat inc(pocet2) until vstup5;
  rychlost:=delka/1000/pocet2*pocet1;
  gotoxy(1,3);
  writeln(rychlost:10:2,' m/s');
 until keypressed;
 if readkey=#0 then readkey;
end.
```

Uvedený program funguje s autodráhou dobře, avšak obsahuje jednu logickou chybu, resp. zjednodušení. Kvůli němu měří trochu nepřesně, což u autodráhy asi nikdy nezjistíme a neověříme. Vycházeli jsme totiž z toho,

že kalibrační a měřící smyčka jsou stejné, ale ony nejsou. Kalibrační končí vyhodnocením podmínky ' ... until not(vstup6); ' zatímco měřicí podmínkou ' ... until vstup5 '. Že jde o různé vstupy, to nevadí, ale ta negace v prvním případě se na rychlosti smyčky nepatrně projeví. Kalibrační smyčka je pomalejší, naměříme tedy větší rychlost, než je skutečná. Navíc tím, že každé použití vstup6 nebo vstup5 je voláním procedury, ztrácíme další čas. Smyčka by mohla být ještě podstatně rychlejší, stačilo by přepsat podmínky přímo bez volání procedur.

Automatické řízení

Některé autodráhy jsou vybaveny takzvaným automatem, což je vzletný název pro reostat, kterým se nastaví na jedné dráze stabilní napájení tak, aby auto jelo co nejrychleji a přitom v žádné zatáčce nevyletělo. Je určen především dětem, které si hrají s autodráhou samy a takto dostanou možnost závodit, i když jen proti „automatu".

Podstatně dokonalejší automat si můžeme naprogramovat. Závodní dráhu si rozdělíme na několik úseků, které se liší rychlostí auta. Delší rovný úsek dovoluje jet naplno, vertikální smyčka se přímo musí projet plnou rychlostí, naopak do zatáček je třeba ubrat, jinak při jízdě na vnitřní a jinak na vnější dráze. Máme k dispozici šest vstupů, můžeme tedy rozdělit dráhu až na šest dílů.

Nejjednodušší příklad dráhy je na obr. 22. V místech označených šipkami jsou umístěna čidla (vstupy 4 a 5), auto je napájeno ze zdroje +5 V (tento typ autodráhy je určen na 4,5V) přes spínač řízený impulzním signálem s proměnnou střídou z D/A převodníku p1 až p4. Kmitočet spínání je lepší nižší, něco mezi 20 a 50 Hz.

Nejprve postupně zvyšujeme rychlost a najdeme takovou, kdy auto bezpečně projede dráhu. Tuto rychlost nastavíme potom pro úsek se zatáčkami „A". Pro úsek „B" nastavíme plný výkon. Protože však auto má svou setrvačnost (nezrychlí hned a také hned nezpomalí), musíme posunout čidla o kousek proti směru jízdy tak, aby auto akcelerovalo již při výjezdu ze zatáčky a naopak ubralo již v předstihu před další zatáčkou.

Výchozí program je velmi jednoduchý. Při hledání rychlosti pro úsek „A" nejprve nastavujeme obě konstanty stejně.

```
Program autopilot1;
{pr_025.pas}
{zkusebni autopilot pro oval}
uses ovl,crt;
const rychlost_a=6;
      rychlost_b=15;

begin
 init;
 {rozjezd na start}
 rychlost_1(rychlost_a);
 repeat
  repeat until vstup4;
  {budou zatacky}
  rychlost_1(rychlost_a);
  repeat until vstup5;
  {bude rovinka}
  rychlost_1(rychlost_b);
 until keypressed;
 rychlost_1(0);
 if readkey=#0 then readkey;
end.
```

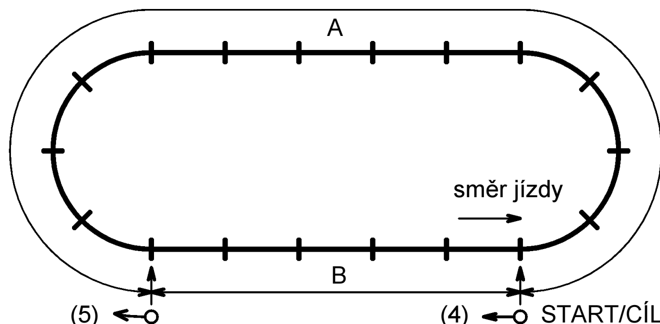
S více čidly je možné si potom opravdu vyhrát. Na obr. 23 je příklad skutečné dráhy, rozdělené do šesti úseků „A" až „F". Úsek "A" je z pohledu autopilotem řízeného vozu vnější zatáčka, úsek „B" vnitřní zatáčka se stoupáním na most, úsek „C" je rovné klesání, na které navazují dvě křížení v úseku „D". „E" je opět vnější zatáčka a poslední úsek „F" je dlouhá rovinka před cílem. Každý úsek má jinou maximální rychlost (kromě „A" a „E", ty mají stejný charakter), pro každý úsek ji musíme najít a nastavit.

Program by měl kromě řízení rychlosti umožňovat i měřit čas na jedno kolo, vždyť to je konec konců rozhodující. Výsledek není nijak složitý, jen hledání optimálního nastavení je pracné a zdlouhavé - ale dětem to tak vůbec nepřipadá. Samozřejmě, že každé auto má své vlastní trochu jiné nastavení autopilota, protože má i jiné vlastnosti v zatáčkách. Součástí optimalizace je i posunutí snímačů proti směru jízdy stejně jako v minulém případě.

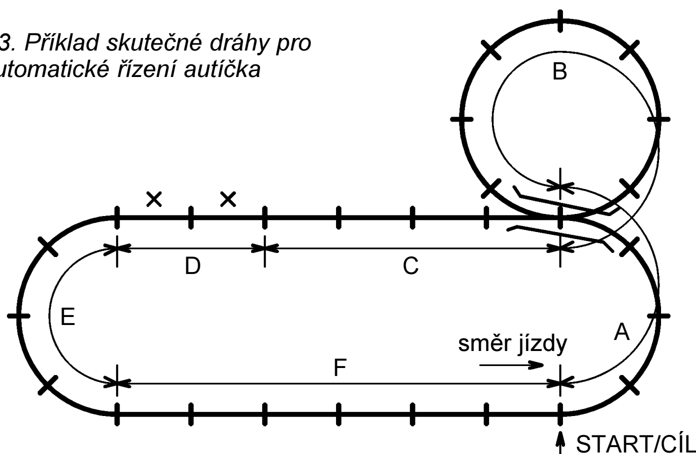
```
Program autopilot2;
{pr_026.pas}
{autopilot pro vetsi drahu}
uses ovl,crt;
const rych_a=12;
      rych_b=14;
      rych_c=10;
      rych_d=7;
      rych_e=9;
      rych_f=15;
var cas1,cas2:real;
 kolo:word;

procedure zastav;
 begin
 if keypressed then begin
  if readkey=#0 then readkey;
  rychlost_1(0);
  exit;
  end;
 end;

begin
 init;clrscr;
 kolo:=0;
 rychlost_1(10);
 writeln('Autopilot pro autodrahu');
 writeln;
 repeat
  repeat zastav until vstup1;
  vystup_p(rych_a);
  cas2:=cticas;
  if kolo>0 then begin
   write(' Kolo ',kolo:5,' : ');
   writeln((cas2-cas1):10:2,' s');
   end;
  inc(kolo);
  cas1:=cas2;
  repeat zastav until vstup2;
  rychlost_1(rych_b);
  repeat zastav until vstup3;
```



Obr. 22. Nejjednodušší příklad dráhy pro automatické řízení autíčka



Obr. 23. Příklad skutečné dráhy pro automatické řízení autíčka

```
   rychlost_1(rych_c);
   repeat zastav until vstup4;
   rychlost_1(rych_d);
   repeat zastav until vstup5;
   rychlost_1(rych_e);
   repeat zastav until vstup6;
   rychlost_1(rych_f);
  until false;
 end.
```

Vyladit vhodné konstanty ve funkčním programu trvá asi půl až jednu hodinu. Výsledkem je však „závodník”, kterého není snadné porazit. Jede totiž strojově přesně, nedělá chyby a rychlost řídí citlivěji, než je většinou možné ručním ovladačem a s rychlostí reakce člověka.

Program lze dále zdokonalovat. Rychlost auta je řízena střídou impulzů, ale stejně výrazně závisí i na jejich napětí. Pokud toto napětí nemá vždy stejnou velikost, vyplatí se zavést konstantu - koeficient většinou blízký jedné - který při použití ovlivní poměrným způsobem všechny ostatní rychlostní konstanty. Lze tak jednoduše kompenzovat změnu napětí.

Protože každé auto má své vlastní optimální nastavení, přímo se nabízí toto nastavení ukládat do souboru a opět z něj načíst (vytvořit vlastně knihovnu aut pro danou dráhu) a knihovnu různých tvarů dráhy.

Dále lze zapsat do konstant požadované rychlosti (časy) v jednotlivých úsecích dráhy, měřit skutečnou rychlost (čas) dosaženou v těchto úsecích a podle toho upravovat rychlost v dalším kole pro každý kus dráhy samostatně.

Je také možné (a ani ne moc složité) vytvořit celý učící se systém, který sám dokáže i zcela neznámou dráhu zmapovat, nalézt její optimální nastavení a to potom uložit do souboru pro další použití. Podmínkou je, aby rozmístění čidel podél dráhy bylo zvolené rozumně, to program nezkontroluje. Princip je celkem jednoduchý.

Program na počátku neví nic o délkách jednotlivých úseků a jejich charakteru. Může však auto rozjet nízkou rychlostí a změřit časy všech úseků. Pak trochu (o jeden stupeň) přidá, znovu měří. Jednotlivé časy by měly klesnout. Jestliže některý z nich výrazně vzroste (a v praxi je to vzrůst na několikanásobek, nejen o trošku), program tím pozná, že auto vypadlo z dráhy a pro daný úsek rychlost v dalším kole zpátky sníží.

Tak postupně program najde maximální rychlost pro všechny úseky. Optimalizace končí, když auto řízené podle nalezených a víceméně už stabilních hodnot rychlosti objede celý okruh řekněme pětkrát bez vylétnutí z dráhy. Po celou dobu hledání a nastavování nemusí člověk dělat nic jiného, než občas sebrat auto a nasadit ho zpět na dráhu ve stejném úseku, v kterém dráhu opustilo. Ani se nemusí moc spěchat, bylo by to spíš ku škodě věci.

Proces vyžaduje určitý čas, auto objede dráhu při optimalizaci několikrát i stokrát. Zato soupeřit s ním potom na sousední dráze, to dá opravdu zabrat.

Doplňky k modelové železnici

Na první pohled by se mohlo zdát, že modelová železnice a autodráha mají z hlediska ovládání mnoho společného. Není tomu tak. Autodráha vystačí pouze s řízením rychlosti při jízdě vpřed, jde především o závodění. U železnice je změna směru jízdy běžná. Můžeme ovládat výhybky, měnit jízdní trasu, rozpoznávat osobní a nákladní vlaky jedoucí po stejné koleji a různým způsobem je obsluhovat. Především však modelová železnice nejsou jen vlaky na kolejích. Podstatně více ovladatelných doplňků najdeme kolem trati.

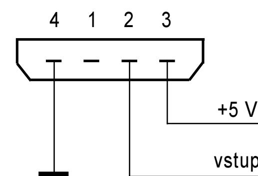
V napájení modelové železnice není úplná jednotnost. Lokomotivy jsou zpravidla konstruovány na stejnosměrné napětí 12 V, vyhybky a návěstidla vyžadují obvykle 12 až 16 V střídavých (50 Hz), ale často jsou schopny fungovat i na stejnosměrné napětí 16 až 18 V. Je dobré si ověřit, že použité výhybky odpojují napájení cívek v koncových polohách, v opačném případě by se snadno mohly trvalým proudem zničit.

Modelová železnice se snaží přiblížit vzhledem i funkcí k realitě. O totéž se budeme snažit i my. Jen příčiny a důsledky určitých dějů si občas dovolíme obrátit. Například ve skutečnosti se vlaky řídí podle návěstidel. My můžeme někdy ovládat návěstidla podle pohybu vlaku.

Pro snímání polohy soupravy je použití optických čidel problematické. Zdroje světla ani čidla by neměla být vidět nebo by alespoň neměla být nápadná, snímací „most” jako u autodráhy prakticky nepřichází v úvahu. Především však mezerami mezi vagóny světlo určitě projde a s tím musíme počítat. Snadno zaregistrujeme začátek vlaku, ale jeho konec musíme vyhodnocovat složitěji.

Jednou z možností, jak snížit nápadnost, je používat infračervená čidla (např. L-NP-3C1) a infračervené LED (např. L-NP-7C1) jako zdroj světla pro ně. Oba uvedené typy jsou velmi ploché a svítí (přijímají) ve směru kolmo na plochu, vejdou se snadno i mezi dvě sousední koleje a dají se celkem lehce zamaskovat.

Můžeme využít jazýčkové kontakty zatavené do skleněného pouzdra a vlepené mezi koleje. Na podvozek lokomotivy nebo vagónů pak nalepíme miniaturní magnet, který při průjezdu kontakt sepne. Tak lze velmi jednoduše odlišit třeba nákladní vlaky od osobních - jedny mají magnet, druhé ne. Zapojení kontaktu je jednoduché - mezi příslušný vstup a kladný pól napájecího napětí 5 V.



Obr. 24. Zapojení vývodů Hallovy sondy typu MH1SS1

Modernější (ale složitější) obdobou magnetického spínače jsou snímače na principu Hallových sond. Starší české typu MH1SS1 (připojení viz obr. 24) a podobné najdeme v bazarech nebo vyřazených klávesnicích, nové a velmi malé v provedení SMD lze získat rozebráním mechaniky vyřazené disketové jednotky. V obchodech se součástkami se shánějí špatně.

Pískáček

V některých úsecích najdeme u trati červenobíle pruhovaný kolík, kterému se říká pískáček. Je to návěstí pro strojvůdce, že má dát přerušované zvukové znamení. Parní lokomotivy dříve pískaly, proto pískáček. Dnešní lokomotivy houkají hlubokým tónem. Zkusme zařídit totéž na svém jednoduchém kolejišti.

V určité vzdálenosti od nechráněného přejezdu umístíme pískáček. O pár centimetrů dál od přejezdu instalujeme optické čidlo průjezdu vlaku (vstup 2) a dalších pár centimetrů dál mezi koleje čidlo magnetické (vstup 1). Na podvozek parních lokomotiv přichytíme malý magnet, který je schopen čidlo sepnout. Ostatní lokomotivy magnety mít nebudou.

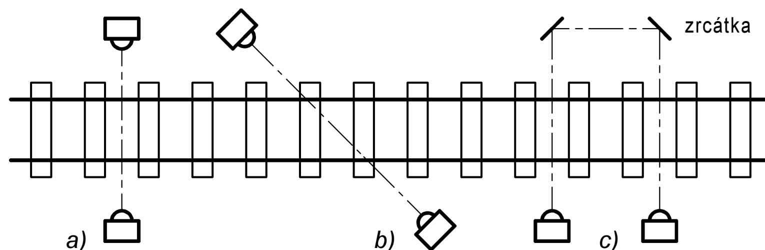
Je třeba brát v úvahu, že vlak může na přejezd vjet lokomotivou napřed nebo nacouvat (vagóny napřed). Pokud chceme i couvat, musíme umístit magnet i na poslední vůz parní soupravy. Pro jednoduchost zatím uvažujeme jen jednosměrnou kolej.

Funkce bude jednoduchá. Jakmile se vlak přiblíží, sepne magnetické čidlo - tedy pokud je to vlak parní. Vzápětí indikuje přítomnost vlaku optické čidlo, které spustí zvukovou výstrahu - pískání nebo houkání. Víc není nutné. Zvuk však budeme generovat trochu jinak než dosud. Použijeme unitu COMINT a výstup TxD třeba portu COM2. TxD připojíme na vstup univerzálního posilovače a na jeho výstup přes rezistor o odporu 10 Ω dáme malý reproduktorek - výkon ani impedance není příliš důležitá.

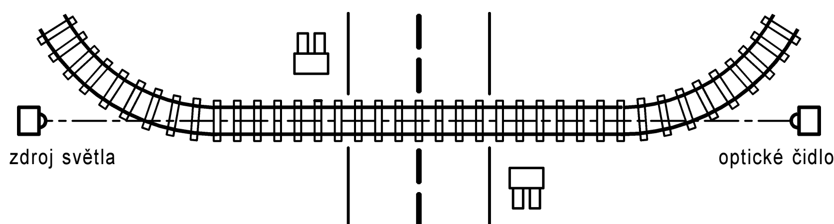
```
Program piskacek;
{pr_027.pas}
{zvukove znameni pred prejezdem}
uses crt,ovl,comint;
var cas:integer;
 parni:boolean;

begin
 init; randomize; cas:=0;
 repeat
  if vstup1 then begin
```

```
    parni:=true;
    cas:=500;
    end;
   if vstup2 then begin
    if parni then initint(2,1300)
     else initint(2,140);
    cekej(random(6)+3);
    deinitint;
    cekej(random(4)+2);
    if parni then initint(2,1300)
     else initint(2,140);
    cekej(random(12)+5);
    deinitint;
    cekej(random(11)+9);
    end;
   dec(cas);
   if cas<0 then begin
    cas:=0;
    parni:=false;
    end;
   delay(10);
  until keypressed;
  if readkey=#0 then readkey;
 end.
```



Obr. 25. Umístění optických čidel u kolejí. a) - kolmý paprsek, b) - šikmý paprsek, c) - 2x odražený paprsek



Obr. 26. Umístění optického čidla na rovném úseku kolejí mezi zatáčkami

Aby zvukové znamení, které ani v drážních předpisech nemá přesnou délku, více připomínalo skutečnost, je vždy náhodným generátorem mírně pozměněno.

Rozšíření programu je nasnadě. Přidáme další dvě čidla tak, aby indikace správně fungovala i při obousměrném provozu. Nesmíme zapomenout, že po průjezdu přejezdem se už nepíská, i když druhé čidlo samozřejmě sepne také.

Kdybychom chtěli přesně dodržet traťové předpisy, bude vlak vydávat přerušované zvukové znamení od okamžiku průjezdu kolem pískáčku až do chvíle, kdy jeho čelo projede kolem přejezdu.

Signalizace na přejezdu

Běžná situace z našich silnic. Světelná signalizace na přejezdu s bílým blikajícím světlem, při přiblížení vlaku se střídavě rozblikají dvě červená světla a je slyšet cinkání zvonku. Červená světla uzavírají přejezd až do doby, než celý vlak projede. Vyrobíme si totéž, opět pro začátek budeme uvažovat jen jednosměrnou trať.

Asi 50 cm před přejezd umístíme optické čidlo průjezdu vlaku, stejné druhé čidlo dáme asi 10 cm za přejezd. Čidla jsou připojena na vstupy 1 a 2. V klidu pomalu bliká bílé světlo (LED na výstupu 1). Jakmile čidlo před přejezdem zaregistruje vlak, zhasne bílé světlo a střídavě se rozblikají světla červená (výstupy 2 a 3). Současně se ozývá i zvukový signál. S věrností napodobení zvonku si starosti dělat nebudeme, krátké pípnutí stačí. Zvuk budeme generovat stejně jako v předchozím případě. Potom čekáme, až vlak dojede k čidlu za přejezdem a celý kolem něj přejede.

```
Program signalizace;
{pr_028.pas}
{zvukove znameni pred prejezdem}
uses crt,ovl,comint;
var vlak,stav,posled,konec:boolean;
 i:integer;

function prujezd:boolean;
 begin
 stav:=vstup2;
 if (not stav) and posled then
  prujezd:=true else prujezd:=false;
 posled:=stav;
 delay(1);
 end;

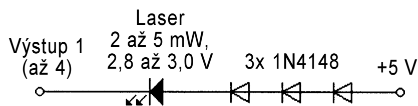
begin
 init;
 vlak:=false;
 posled:=false;
 konec:=false;
 repeat
  if vlak then begin
   zapni2;
   for i:=1 to 800 do if prujezd
    then konec:=true;
   vypni2;
   initint(2,2000);
   delay(10);
   deinitint;
   zapni3;
   for i:=1 to 800 do if prujezd
    then konec:=true;
   vypni3;
   end
  else begin
   zapni1;
   for i:=1 to 600 do begin
    if vstup1 then vlak:=true;
    delay(1);
    end;
   vypni1;
   for i:=1 to 1000 do begin
    if vstup1 then vlak:=true;
    delay(1);
    end;
   end;
  if konec then vlak:=false;
  konec:=false;
 until keypressed;
 if readkey=#0 then readkey;
end.
```

Jestliže vlak na přejezdu zastaví a čidlo se bude dívat právě mezerou mezi vagóny, bude přejezd uvolněn, ačkoli by být neměl. Podobně bude chybně vyhodnocena situace, jestliže lokomotiva dojede až přes druhé čidlo, zastaví a pak začne couvat. Jak tomu zabránit? Možností je několik.

Můžeme zvětšit počet čidel na čtyři ve vzdálenostech od přejezdu -50, -10, +10 a +50 cm. Tato čidla se stejně hodí, až budeme rozšiřovat funkci pro obousměrný provoz. Potom sledujeme všechna čtyři čidla - vlak musí nejen svým začátkem čidla postupně aktivovat, ale i po projetí je uvolnit v daném pořadí. Pokud se to nestane, vlak zastavil nebo dokonce ve sledovaném úseku couval. Vyhodnocení situace se dost zkomplikuje.

Ne vždy je programové řešení to nejjednodušší a nejspolehlivější, existují i jiné varianty. Třeba upravit snímací čidla podle obr. 25, aby mezerou mezi vagóny prostě neviděla. Světlo neprochází kolmo na koleje (obr. 25a), ale šikmo pod úhlem asi 30° (obr. 25b). Tak nemůže projít mezerou mezi vagóny. Velmi důležitá je samozřejmě i výška nad kolejemi, ve které paprsek prochází. Nízký plošinový vagón ho nesmí podjet. Optimální výška nad kolejemi je asi rovna průměru kol. Další možností je nechat paprsek dvakrát odrazit, tedy dvakrát přejít koleje ve vzdálenosti, která je o něco větší než nejširší mezera mezi vagóny (obr. 25c). Ani v tomto případě čidlo mezeru nezaregistruje. Tím se však zbavíme jen části možných problémů.

Pokud je přejezd umístěn v zatáčce nebo její blízkosti, a to na omezeném prostoru modelových kolejišť platí skoro vždy, můžeme se situaci ještě zjednodušit. Jako zdroj světla použijeme žárovku a čočkou, abychom dostali úzce směrový zdroj. Paprsek necháme procházet nad delším úsekem kolejí mezi zatáčkami (obr. 26). Jestliže je přerušen, je vlak v úseku a bez



Obr. 27. Zapojení laserové diody

ohledu na to, co dělá a jakým směrem jede, musí svítit výstražná světla. Tento způsob je na obsluhu programem zdaleka nejjednodušší, má však i svou nevýhodu. V šeru bude paprsek osvětlující vlak vidět, světlo musí být kvůli větší vzdálenosti od čidla silnější.

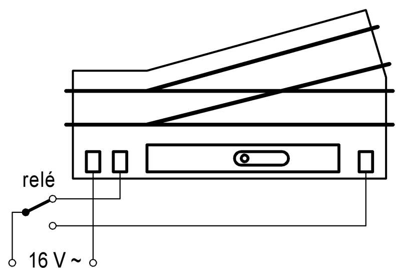
Pokud chceme zvládnout i tuto nevýhodu, místo žárovky jako zdroje světla použijeme vysoce svítivou LED nebo dokonce malý polovodičový laser. Není nijak drahý a včetně pouzdra je i menší, než běžné žárovičky. Nepotřebuje čočku (tu už má v sobě) a hlavně, dají se jím dělat i velmi krátké impulzy světla (to s žárovkou nejde). Počítač pak periodicky zjišťuje přítomnost vlaku v úseku tak, že zapne zdroj světla, zkontroluje stav čidla a hned zase světlo vypne. Tyto krátké záblesky jsou pro oko prakticky neviditelné, postačí 1 ms čtyřikrát za sekundu. Zapojení je na obr. 27.

Automatické závory

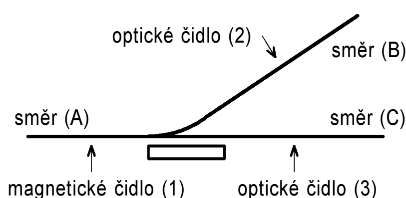
Tento příklad je obdobou předchozího, detekce vlaku už byla vysvětlena v závěru minulého příkladu, odlišnost je ve spínání výstupu. Závory se ovládají stejně jako výhybky střídavým napětím, takže místo blikání červených světel (možná k němu navíc) sepneme pomocí relé 1 střídavý proud pro závory. Uvedeme si zde příklad jednoduché obsluhy závor s laserem (LED) na výstupu 1 a čidlem na vstupu 1.

```
Program zavory;
{pr_029.pas}
{stahovani zavor, detekce laserem}
uses crt,ovl;

begin
 init;
 repeat
  zapni1;
  delay(1);
  if vstup1 then zapni_re1
   else vypni_re1;
  vypni1;
  delay(499);
 until keypressed;
 if readkey=#0 then readkey;
end.
```



Obr. 28. Ovládání výhybky



Obr. 29. Odbočka na trati

Ovládání výhybky

Výhybky, alespoň ty starší, jsou typicky konstruovány pro střídavé napětí 16 V. Pokud mají v pořádku kontakty, které v koncových polohách vypínají proud do cívek, můžeme je jednoduše ovládat pomocí relé na desce Interface PC, které zapojíme podle obr. 28.

Připravíme si odbočku na trati (obr. 29), kam bude automaticky odkloněn každý nákladní vlak (tj. vlak vybavený magnetem na podvozku lokomotivy), zatímco osobní vlaky pojedou přímo. Asi 20 cm před výhybku umístíme mezi koleje magnetické čidlo připojené na vstup 1. Na odbočku i rovný úsek, asi 10 cm za výhybkou, dáme šikmo čidla optická (vstupy 2 a 3) tak, aby nemohla vidět mezery mezi vagóny. V klidovém stavu je výhybka přepnuta na rovno. Jestliže se blíží nákladní vlak ze směru (A), přehodí počítač výhybku na odbočení a hlídá čidlo za odbočkou. Až poslední vagón vlaku přejede, výhybku opět srovná. Na osobní vlak výhybka nereaguje a vlak projede rovně. Jestliže se blíží jakýkoli vlak ze směru (B), musí přehodit vyhybku a počkat, až projede. Jestliže přijíždí osobní vlak ze směru (C), také se nic neděje. Kritická je však možnost, že ze směru (C) přijíždí nákladní vlak. Jeho magnet by po aktivaci čidla přehodil výhybku právě v době, kdy přes ní jede zbytek soupravy a ta by vykolejila. Proto je optické čidlo i ve směru (C). Jestliže z tohoto směru jede vlak, nesmí se až do jeho průjezdu a určitý čas po něm brát v úvahu hlášení magnetického čidla.

Uvedený postup neřeší všechny situace, třeba tu, kdy dva rozdílné vlaky pojedou velmi těsně za sebou. Běžný provoz na kolejišti však zvládne. Ve většině situací spoléháme na to, že vlak určitě opustí prostor výhybky do určité doby, která je nastavená konstantou. Tím se program značně zjednoduší.

```
Program odkloneni;
{pr_030.pas}
{odkloneni nakladnicch vlaku}
uses crt,ovl;
const doba=80; {x 0.1 s}
var odklon:boolean;

begin
 init;
 odklon:=false;
 repeat
 if vstup2 then begin
  zapni_re1;
  repeat cekej(1) until not vstup2;
  cekej(doba);
  vypni_re1;
  end;
 if vstup3 then begin
  vypni_re1;
  repeat cekej(1) until not vstup3;
  cekej(doba);
  end;
 if vstup1 then begin
  zapni_re1;
  repeat cekej(1) until vstup2;
  repeat cekej(1) until (not vstup2);
  vypni_re1;
  end;
 until keypressed;
 if readkey=#0 then readkey;
end.
```

Rozšířit program můžeme třeba tak, že přidáme třetí optické čidlo ze směru (A), o kousek blíž k výhybce než čidlo magnetické. Ve všech směrech průjezdu vždy zajistíme, aby se výhybka přehodila do původní polohy (nebo byla blokována) až do úplného projetí vlaku. Časová konstanta bude potom zbytečná.

Regulace při rozjíždění a brždění

Regulaci rychlosti pohybu vláčků můžeme řešit téměř stejně jako u autodráhy, určité rozdíly tu však přece jen jsou. Předně motorek lokomotivy reaguje velmi dobře i na nejmenší střídu impulzů a vlak jede při kmitočtu kolem 20 Hz pěkně a velmi pomalu vpřed. To se na autodráze nestává. Vždy budeme uvažovat i možnost změny směru pohybu pomocí relé (obr. 18).

Protože jednotlivé způsoby ovládání už byly probrány dříve, podíváme se na celý problém z jiné stránky. Budeme využívat regulaci rychlosti střídou přes převodník D/A (p1 až p4), jeden univerzální posilovač a relé 1 pro změnu směru.

Jestliže vyzkoušíme, jak lokomotiva ve skutečnosti reaguje na změnu střídy, brzy zjistíme, že zatímco rozdíl při prvních stupních je dobře patrný, s rostoucí rychlostí skoro mizí. Jestliže necháme lokomotivu postupně na každý krok střídy jet vždy jednu sekundu a pozorujeme výsledek, rozjezd je na začátku přirozený, ale později už neznatelný. Přitom trvá příliš dlouho a lokomotiva ujede během něj několik metrů, což je na malé kolejiště moc.

```
 ...
 for i:=0 to 15 do begin
  rychlost_1(i);
  delay(1000);
  end;
 rychlost_1(0);
 ...
```

I když to v podstatě odpovídá skutečnosti (zrychlení z 5 na 10 km/h poznáme snadno, zrychlení z 95 na 100 km/h skoro vůbec), přirozenější a hezčí rozjezdy uděláme i s menším po-

čtem stupňů rychlosti, pokud je vhodně vybereme. Můžeme zkusit třeba stupně 0, 1, 2, 3, 5, 9, 15. Vytvoříme si proceduru rychlostvlaku(byte), jejímž parametrem bude požadovaná rychlost vlaku jako přirozené číslo od 0 do 6. Tuto proceduru pak budeme používat místo dosavadní.

```
...
Procedura rychlostvlaku(r:byte);
begin
 case r of
  0 .. 3: rychlost_1(r);
  4: rychlost_1(5);
  5: rychlost_1(9);
  else rychlost_1(15)
  end;
 end;
...
...
for i:=0 to 6 do begin
 rychlostvlaku(i);
 delay(1000);
 end;
rychlostvlaku(0);
...
```

Druhou možností je regulovat čas, po který jednotlivé stupně jsou udržovány. Budeme zatím vycházet z toho, že vždy jde o rozjezd na plnou rychlost. Následující program vlak z výchozí pozice plynule rozjede, pak udržuje plnou rychlost vlaku až k optickému čidlu připojenému na vstup 1 a poté plynule vlak zastaví. Konstanta zrychleni nastavuje rychlost rozjezdu.

```
Program rozjezd;
{pr_031.pas}
{plynuly rozjezd a zastaveni vlaku}
uses crt,ovl;
const zrychleni=3000;
var i:integer;

begin
 init;
 for i:=1 to 15 do begin
  rychlost_1(i);
  delay(round(zrychleni/i));
  end;
 repeat until vstup1;
 for i:=15 downto 1 do begin
  rychlost_1(i);
  delay(round(zrychleni/i));
  end;
 rychlost_1(0);
end.
```

Jestli použijeme první nebo druhý způsob, je víceméně jedno. Oba vypadají na pohled přirozeně, druhý má o něco plynulejší průběh.

Zastavení ve stanici

Představme si jednoduchou zastávku na jednokolejné (a pro začátek jednosměrné) trati. Osobní vlak se blíží, projede kolem návěstidla, které se přestaví na stůj. Pomalu dobrzdí přesně u začátku nástupiště, počká (otevírat dveře a nechat cestující nastupovat v modelovém měřítku přece jen neumíme). Další návěstidlo mu dá volno, ozve se zapískání a vlak se pomalu rozjíždí. Jakmile opustí stanici, uvolní se opět vjezd do ní. Pokud přijede nákladní vlak, jen zahouká, sníží rychlost a stanicí plynule projede. To vše můžeme dělat bez přímého řízení člověkem.

Prvním krokem ke zvládnutí tohoto úkolu je předchozí příklad. Změříme si, jakou dráhu potřebuje lokomotiva k zastavení řekněme při konstantě zrychleni=1200. Zkusíme jízdu bez zatížení i brzdnou dráhu s připojenými vagóny. Pravděpodobně se délka příliš lišit nebude, nejvýraznější setrvačnost z celé soupravy má lokomotiva. Různé lokomotivy se však budou lišit hodně.

Asi metr před zastávku (resp. před bodem požadovaného zastavení) umístíme první magnetické čidlo (vstup 1). To rozpozná nákladní vlak (s magnetem). Zvukový výstup je na TxD COM2, přes univerzální posilovač a rezistor o odporu 10 Ω se budí reproduktor. Návěstidlo je připojeno na výstupy 1 (zelená) a 2 (červená). Asi 60 cm před bod zastavení umístíme optické čidlo (vstup 2). To určuje začátek brždění osobního vlaku.

Brždění je nastaveno tak, aby vlak plynule zastavil před určeným místem. Vlivem různého zatížení lokomotivy a tření probíhá brždění pokaždé trochu jinak. Proto nebudeme brzdit až do zastavení, ale řekněme jen na stupeň 5 (z 15). Pak budeme držet rychlost až do průjezdu vlaku kolem optického čidla těsně před požadovaným bodem zastavení (vstup 3). Poté plynule dobrzdíme do zastavení.

Čekání ve stanici, zvukový signál píšťalky a opětovný rozjezd je pak už celkem snadnou záležitostí. Před rozjezdem se přestaví další návěstidlo na volno - výstup 3 zelená, výstup 4 červená.



Modelové návěstidlo

Při uvedeném způsobu zastavování se zmenší rozdíly mezi bodem zastavení různých souprav asi na 5 mm. Program v průběhu činnosti vypisuje, co se na trati právě děje.

```
Program zastavka;
{pr_032.pas}
{automaticky provoz zastavky}
uses crt,ovl,comint;
const zrychleni=2500;
 brzdeni=1200;
var i:integer;
 naklad:boolean;

procedure konec;
 begin
 if keypressed then begin
  if readkey=#0 then readkey;
  init;
  rychlost_1(0);
  halt;
  end;
 end;

begin
 init; clrscr;
 zapni1; zapni4; rychlost_1(15);
```



Modelové nádraží

```
naklad:=false;
writeln('Automaticky provoz stanice');
repeat {hlavni smycka programu}
 writeln;
 repeat {cekani na vlak}
  konec;
  if vstup1 then begin
   naklad:=true;
   initint(2,160);
   vypni4;
   end;
 until vstup2; {prijel vlak}
 konec;
 if naklad then
  writeln('Jede nakladni vlak')
   else
  writeln('Jede osobni vlak');
 deinitint;
 vypni1; zapni2;
 if naklad then begin {nakladni
  vlak}
  vypni4; zapni3;
  writeln('  pribrzdi');
  for i:=15 downto 5 do begin
   rychlost_1(i);
   delay(round(brzdeni/i));
   end;
  writeln('  projede stanici');
  repeat konec until vstup3;
  {na konci}
  writeln('  opusti stanici');
  repeat konec until not vstup3;
  {odjel}
  delay(2000);
  writeln('  zrychluje');
  for i:=5 to 15 do begin
  {zrychluje}
   rychlost_1(i);
   delay(round(zrychleni/i));
   end;
  konec;
  zapni4; vypni3;
  delay(1000);
  writeln('  je pryc');
 end else begin {je to osobni
  vlak}
  writeln('  zpomaluje');
  for i:=15 downto 4 do begin
   rychlost_1(i);
   delay(round(brzdeni/i));
   end;
  writeln('  dojizdi k nastupisti');
  repeat konec until vstup3;
  writeln('  zastavuje');
  for i:=3 downto 0 do begin
   rychlost_1(i);
   delay(300);
   end;
  writeln('  stoji');
  konec;
  delay(4000);
  konec;
  zapni3; vypni4;
  delay(2000);
  initint(2,1150);
  writeln('  povel k odjezdu');
  vypni4; {nutne po initint}
  delay(800);
  deinitint;
  konec;
  delay(1000);
  writeln('  rozjizdi se');
  for i:=1 to 15 do begin
   rychlost_1(i);
   delay(round(zrychleni/i));
   end;
  repeat konec until not vstup3;
  writeln('  opustil stanici');
  delay(1000);
  writeln('  je pryc');
 end;
 naklad:=false;
 zapni1; vypni2; zapni4; vypni3;
until false;
end.
```

Tento program je již značně dlouhý a pracný, není však nijak složitý, jen pracuje s více vstupy a výstupy. Na přípravku nám zbyly ještě nevyužité tři vstupy, jeden výstup TxD a kompletní řízení rychlosti a směru jednoho motoru převodníkem (nebo vyhybky pomocí relé).

Úlohu můžeme dále doplnit tak, aby vlak mohl přijíždět k nástupišti z obou směrů. Můžeme vybudovat malé nádraží s druhou kolejí, na kterou zajíždějí a na ní zastavují osobní vlaky, ostatní nádražím jen rovně projedou. Možností je velmi mnoho.

Úprava modelářského serva HS-300

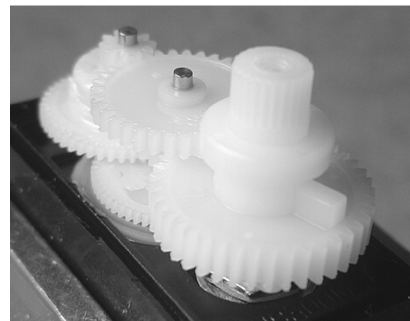
Často se nám při stavbě pokusných konstrukcí a modelů hodí malý stejnosměrný motor na 5 V s převodem do pomala. Nejjednodušším způsobem ho získáme úpravou vyřazeného modelářského serva. Popíšeme si postup pro servo firmy Hitec HS-300, které patří mezi nejlevnější na našem trhu a také je velmi často používané. Toto servo je k úpravě velmi vhodné.

Lze využít i starší, již odložená serva s prodřenou snímací dráhou potenciometru, která pro silné škubání nelze už v RC modelech nasadit. Poškozená serva jsou v modelářských bazarech velmi levná nebo je lze získat i zdarma.

Servo rozebereme vyšroubováním čtyř šroubků a sejmutím spodního krytu. Vyjmeme servozesilovač a odpájíme ho od motoru. Sejmeme horní část krytu převodovky a vyndáme převodová kola. Z hřídele potenciometru stáhneme převodové kolo s unašečem kotouče.

Aby se servo mohlo volně protáčet o 360°, musíme ostrým nožem uříznout zarážku na posledním převodovém kole (obr. 30), která spolu s výčnělky na vnitřní straně horního dílu krytu serva vymezuje mechanické dorazy. Materiál opatrně ukrajujeme plátek po plátku, až je zarážka zcela pryč. Je to snazší a rychlejší než řezání pilkou nebo pilování a také tím nevznikají drobné částečky, které bychom pak museli z ozubených kol pracně čistit.

Malým plochým šroubováčkem rozehneme tři plechové výčnělky, které přidržují odporovou dráhu potenciometru. Po jejich narovnání lze odporovou dráhu vyjmout. Kleštěmi nebo větší pinzetou povolíme matku potenciometru a vyjmeme tělo potenciometru ze serva. Zde najdeme plechovou zarážku, která omezuje pohyb potenciometru a brání jeho protočení dokola.



Obr. 30. Převod serva

Nastavíme unašeč do takové polohy, aby zarážka byla přístupná. Posadíme potenciometr na čelisti většího svěráku závitem dolů. Neupínáme silou, poškodil by se závit. Nasadíme na zarážku břit staršího kovového šroubováku a několika údery kladívkem zarážku promáčkneme tak, aby její plech vyplnil otvor v pouzdře potenciometru. Hřídel se musí zcela volně protáčet dokola.

Nyní můžeme začít se zpětnou montáží. Nasadíme potenciometr a připevníme ho matkou. Sestavíme převodovku a nasadíme horní díl krytu. K vývodům motoru připájíme asi 1 m dlouhou tenkou dvojlinku, na kterou předem navlékneme pryžovou průchodku z původního kabelu. Nakonec uzavřeme krabičku serva a zašroubujeme čtyři šroubky v rozích.

Takto upravené servo se napájí napětím 5 V, při tomto napětí vykoná přibližně 1 otáčku za sekundu a lze zatížit momentem do 30 N/cm. K upevnění serva při konstrukci ze stavebnice Merkur nebo podobné poslouží patky pro šrouby. Pokud upravené servo potřebujeme zkombinovat s Legem nebo jinou plastovou stavebnicí, je nejsnazší přilepit ho oboustranné lepící páskou za bok k dílům stavebnice.

Jako mechanický výstup serva se nejvíce osvědčily původní polyamidové kotouče, do kterých provrtáme otvory o průměru 3 nebo 3,5 mm na uchycení šroubky.

Pro pokusy a krátkodobou zátěž (řekněme do 10 minut), po které má servo čas na vychladnutí, výborně vyhovuje. Je kompaktní, dobře se upevňuje a vypadá profesionálně. Jestliže bychom ho chtěli zatěžovat dlouhodoběji (třeba jako pohon), bylo by třeba nechat kryt serva otevřený kvůli chlazení servozesilovače i motoru.

Úpravy jiných typů serv

Na obousměrný servopohon lze upravit mnoho druhů modelářských serv, nejčastěji asi budeme vycházet z opotřebovaných a pro modelářské účely již nepoužitelných kusů.

V první řadě zkontrolujeme po rozebrání převodovky, zda má její poslední kolo (spojené s unašečem výstupního kotouče) ozubení po celém obvodě. U některých typů totiž byly mechanické dorazy otáčení realizovány tak, že na části plastového kola nebyly zuby. Taková serva prakticky upravit nejde.

Musíme najít a odstranit mechanický doraz převodovky. To jde většinou celkem snadno a nevyžaduje to ani speciální nástroje.

Další mechanický doraz se většinou skrývá ve snímacím potenciometru. U některých typů lze potenciometr vyjmout, aniž se tím naruší činnost převodovky - unašeč je uložen v třecím nebo kuličkovém ložisku a na potenciometr se žádné větší síly nepřenáší. To je nejjednodušší varianta.

Ve většině levnějších serv je však potenciometr součástí mechanické konstrukce převodovky a v podstatě tvoří spodní uložení unašeče. Pak musíme obdobně jako u typu HS-300 potenciometr rozebrat, odstranit mechanickou zarážku a také odporovou dráhu nebo kontakty.

Serva, která mají popraskaná převodová kola, chybějící zuby v převodovce, velkou vůli na výstupním unašeči nebo se jim zasekává motor, se raději ani nepokoušíme upravovat, byla by to velmi pravděpodobně ztráta času. Tyto vady jsou typické pro serva zničená při velkých haváriích. Naopak serva se zničenými servozesilovači nebo vydřenými potenciometry (vady typické pro opotřebení dlouhou službou) jsou pro úpravu vhodná.

Řízení neupraveného modelářského serva

Modelářské servo můžeme využít k rozhýbání hraček i v původním stavu. V tom případě je napájecí napětí (které by mělo být přítomno trvale) 5 V, polohu výstupního kotouče serva řídíme krátkými impulzy na signálovém vstupu. Typická výchylka serva je ±90 stupňů od střední (neutrální) polohy a odpovídá vstupním napěťovým impulzům délky 1,00 až 2,00 ms. Opakovací kmitočet impulzů kolem 50 Hz není nijak kritický, na výstupní výchylku má jen malý vliv. Pokud vstupní impulzy ustanou, servo zůstane v dosažené poloze, ale tuto polohu neudržuje (dá se přestavit mechanicky „ručně", mohou se tím však poškodit převody serva).

Impulzy zavádíme do serva přímo z výstupu 1 přípravku Interface PC, rezistor o odporu 2,2 kΩ, spojený s tímto výstupem, připojíme ke kladnému pólu napětí +5 V.

Generovat impulzy o šířce 1 až 2 ms je s využitím čekacích procedur prakticky nemožné, snad jen pokud bychom chtěli využít pouze krajní polohy serva.

```
...
repeat
 vypni1;
 delay(1); {nebo delay(2)}
 zapni1;
 delay(19);
until keypressed;
...
```

K řízení serva využijeme rychlou programovou smyčku, která se musí nejprve na daném počítači zkalibrovat (obdobně jako v příkladu pr_024). Propojíme výstup TxD portu COM1 se vstupem 1. Vygenerujeme na TxD impulz o délce přesně 0,1 s (odpovídající dělitel je 1280) a změříme počet průchodů smyčkou. Jedné krajní poloze serva pak bude odpovídat jedna setina naměřeného počtu (ta současně udává i počet poloh, které může servo mezi krajními polohami zaujmout), druhé poloze dvě setiny.

Zkušební program provede kalibraci, nastaví jednu krajní polohu serva, setrvá v ní 10 s, pak pomalu přejede do druhé krajní polohy a opět v ní setrvá 10 s.

Za povšimnutí stojí dvě věci. Jsou to v první řadě procedury DI a EI v proceduře „pulz", které podstatně zmenšují chvění serva, způsobené neustálým drobným narušováním generování impulzů výkonem přerušení. Pokud tyto procedury odstraníme, nepodstatně se změní impulzy, ale záškuby serva budou velmi dobře pozorovatelné.

Druhou věcí je na první pohled nesmyslná část programu v měřicí smyčce (řada INC(pom)). Má svůj význam. Měřicí smyčka končí totiž podmínkou „not(vstup1)", která je vyhodnocena nesrovnatelně rychleji než podmínka v proceduře pulz, která porovnává dvě proměnné typu word. Zdánlivě zbytečné instrukce kompenzují časování smyček tak, aby trvaly prakticky stejně dlouho.

```
Program servo1;
{pr_033.pas}
{rizeni serva programovou smyckou}
uses ovl,crt,comint;
var min,max,i,pom:word;
 pocet:longint;

procedure pulz(delka:word);
var d:word;
 begin
  d:=0;
  DI;
  vypni1;
  repeat inc(d) until d>=delka;
  zapni1;
  EI;
  delay(18);
 end;

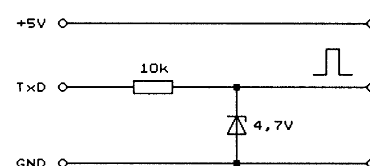
begin
 init;
 {nastaveni parametru portu}
 port[com1+3]:=128;
 port[com1+0]:=lo(1280);
 port[com1+1]:=hi(1280);
 port[com1+3]:=3;
 port[com1]:=0;
 pom:=0;
 {kalibrace podle rychlosti PC}
 pocet:=0;
 repeat until vstup1;
 repeat
  inc(pocet);
  inc(pom); inc(pom); inc(pom);
  inc(pom); inc(pom); inc(pom);
  until not(vstup1);
 min:=round(pocet/10);
 max:=min*2;
 {vychozi poloha}
 for i:=1 to 500 do pulz(min);
 {prejezd}
 for i:=min to max do pulz(i);
 {koncova poloha}
 for i:=1 to 500 do pulz(max);
 pip;
end.
```

Tento způsob řízení serva je opět velmi náročný na čas počítače. Může však mít velké rozlišení, zejména na rychlejších počítačích. Vždy se projevuje alespoň slabé chvění. Stabilnější způsob generování řídicích impulzů vede přímo na využití výstupu TxD v „monostabilním režimu". Tím je míněno, že každý jednotlivý impulz budeme spouštět programově, ale jeho šířka bude vytvořena obvodem UART.

Servo připojíme k výstupu TxD portu COM1 na přípravku Interface PC přes stykový obvod, jehož schéma je na obr. 31. Stykový obvod upravuje řídicí impulzy pro servo tak, aby byly kladné a nepřesahovaly úroveň 5 V, jinak by se servo zničilo. Jinou, rovnocennou možností, je nechat impulzy z TxD projít postupně dvěma univerzálními posilovači (dvěma proto, aby impulzy nebyly invertovány).

Na sériovou linku budeme odesílat byte $FF, který vytvoří kladný impulz v délce jednoho bitu (startbitu). Pro začátek musíme spočítat přenosovou rychlost (resp. dělitele) tak, aby délka byla 1 ms (resp. 2 ms). Doba potřebná k přenesení jednoho bitu je podíl dělitele a čísla 115200, z toho se vypočte dělitel = 0,001 [s] x 115200 = 115 pro impulz o délce 1 ms, pro impulz o délce 2 ms je dělitel roven 230.

Změnou přenosové rychlosti při stejném odesílaném byte dosáhneme změnu šířky impulzu v 115 krocích, což je velmi dobrý základ pro (téměř) plynulé řízení polohy. Následující program zkouší servo stejně jako předchozí, ale v klidové poloze by mělo chvění zcela zmizet. Pokud servo slabě vrčí a nehýbe se, je to způsobeno



Obr. 31. Stykový obvod pro servo. Levé svorky jsou připojeny k přípravku Interface PC, k pravým svorkám je připojeno servo

nedostatečným zesílením zpětné vazby servozesilovače. Motor nedojel až na požadovanou polohu a je napájen krátkými impulzy, které nestačí uvést servo do pohybu. Tento způsob řízení je nezávislý na rychlosti použitého počítače.

```
Program servo2;
{pr_034.pas}
{rizeni serva pres TxD}
uses ovl,crt,comint;
var i:word;

procedure pulz(delka:word);
 begin
  port[com1+3]:=128;
  port[com1+0]:=lo(delka);
  port[com1+1]:=hi(delka);
  port[com1+3]:=3;
  port[com1]:=$FF;
  delay(20);
 end;

begin
 init;
 {vychozi poloha}
 for i:=1 to 500 do pulz(115);
 {prejezd}
 for i:=115 to 230 do pulz(i);
 {koncova poloha}
 for i:=1 to 500 do pulz(230);
 pip;
end.
```

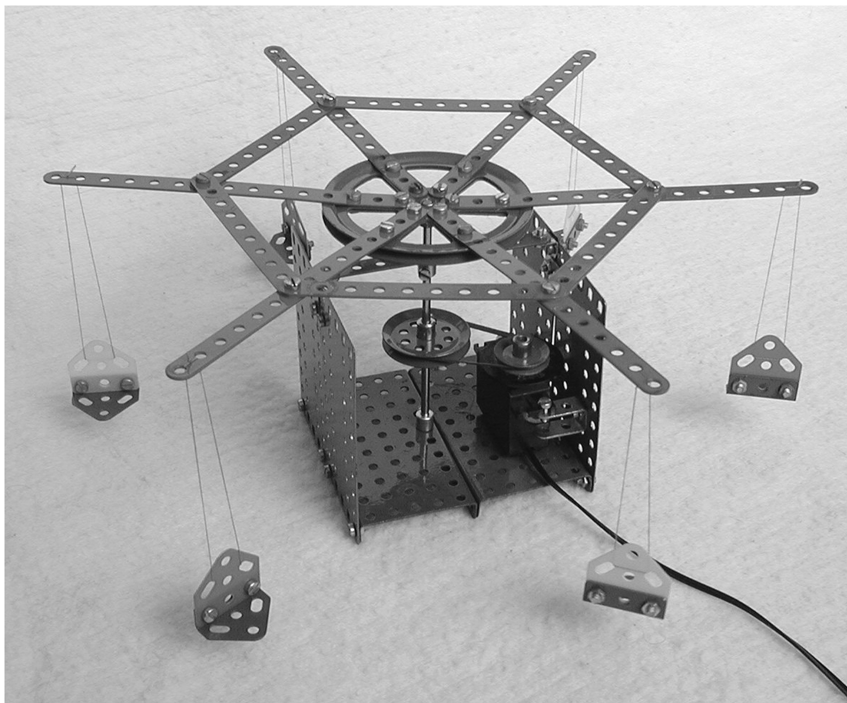
Uvedený program zlepšil ovládání serva, ale zůstal v půli cesty. Stále musíme každý impulz odeslat a hlídat opakovací interval 20 ms. Dalším zlepšením je zavěsit toto odesílání byte na časový interrupt v počítači, který má právě kmitočet 50 Hz. Tím se obsluha serva zjednoduší na zavolání procedury, která upraví nastavení portu pouze v okamžiku, kdy se má poloha serva změnit.

Kolotoč

Kovová stavebnice Merkur (nebo jiný podobný typ, například Meccano) dává široké možnosti ke konstrukci různých pohyblivých hraček. K mechanickému pohonu jsou nejvýhodnější upravená i neupravená modelářská serva.

Velmi jednoduchým příkladem je kolotoč. Jeho funkce bude pouze v tom, že se na povel z klávesnice pomalu postupně roztočí nebo zastaví. Rychlost otáčení postupně zvětšujeme klávesou ↑ (kurzor nahoru) a zmenšujeme klávesou ↓ (kurzor dolů). Rychlost lze nastavit v 16 stupních, aktuální stav je zobrazen na monitoru pomocí řady hvězdiček. Program končí po stisku mezerníku.

Popis mechanické stavby kolotoče je snad zbytečné uvádět, ostatně záleží i na použité stavebnici a dostupném množství jednotlivých dílů. Jako inspirace může sloužit obr. 32.



Obr. 32a. Kolotoč

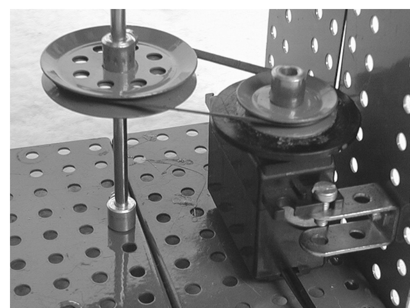
```
Program kolotoc;
{pr_035.pas}
{jednoduchy kolotoc z Merkuru}
uses ovl,crt;
var i,j:integer;
 kl1,kl2:char;

procedure zobraz;
 begin
  gotoxy(20,8);
  write('Rychlost kolotoce   ');
  for j:=1 to i do write('*');
  for j:=(i+1) to 15 do write('.');
 end;

begin
 init; clrscr;
 rychlost_1(0);
 i:=0;
 zobraz;
 repeat
  repeat until keypressed;
  kl1:=readkey;
  if kl1=#0 then begin
   kl2:=readkey;
   if kl2=#72 then inc(i);
   if i>15 then i:=15;
   if kl2=#80 then dec(i);
   if i<0 then i:=0;
   rychlost_1(i);
   zobraz;
   end;
 until kl1=' ';
 rychlost_1(0);
end.
```

Pásový dopravník

Další konstrukcí může být pásový dopravník s podobnou funkcí, jako bývá u pokladny v supermarketu. V klidovém stavu dopravník jede. Položíme-li na pás předmět, bude odvezen až na konec pásu, kde je optické čidlo. Pás se zastaví. Jakmile předmět odebereme, pás se znovu rozjede a přiveze další.



Obr. 32b. Detail pohonu kolotoče

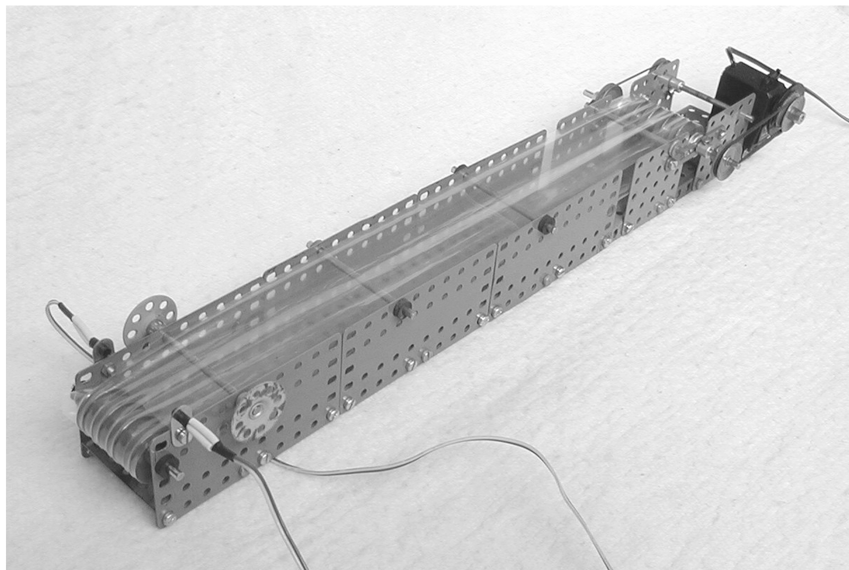
Pohon pásu je tentokrát spínán pomocí relé 1 (není k tomu technický důvod, ale jeho klapání se líbí), optické čidlo je na vstupu 1.

Stejně jako v předchozím případě poslouží místo návodu na mechanické řešení obrázek (obr. 33). Pohon je upraveným modelářským servem. Konstrukce se musí upravit podle pásu, který seženeme - ten na snímku vznikl ustřižením proužku ze silnějšího plastového pytlíku.

Další rozšíření resp. obměna úlohy může spočívat počítání předmětů, jejich dávkování (pás shodí jeden předmět každých 15 s) nebo třeba třídění a počítání předmětů podle velikosti (dáme čtyři optická čidla po 5 mm nad sebe a vyhodnocujeme velikost resp. výšku předmětů).

```
Program dopravnik;
{pr_036.pas}
{pasovy dopravnik}
uses ovl,crt;

begin
 init;
 repeat
  if vstup1 then vypni_re1
   else zapni_re1;
```



Obr. 33. Pásový dopravník

```
   cekej(1);
  until keypressed;
  if readkey=#0 then readkey;
  vypni_rel;
 end.
```

Radar

Zajímavou úlohou je konstrukce „radaru", který vyhledává a zaměřuje zdroj světla. Postavíme model radaru, jehož podstavec tvoří ukotvení pro tělo neupraveného modelářského serva v poloze kotoučem nahoru (výstup TxD COM1 a přes oba univerzální posilovače jako v programu „servo2"). Přímo na kotouč přišroubujeme typickou radarovou anténu, nesmí být však příliš velká a těžká, aby nebylo servo při pohybu namáháno.

Na anténu přichytíme dvě optická čidla (vstup 1 a 2), stejná, jako jsme dosud používali. Fototranzistory by měly být zapuštěny do tmavých trubiček. Osy čidel musí směřovat trochu do stran od nasměrování antény, osvědčil se úhel kolem 25°. Model zkoušíme v místnosti s tlumeným světlem.

Pokud ani jedno čidlo nevidí zdroj světla, anténa přejíždí z kraje do kraje - vyhledává. Jakmile posvítíme na radar a ten zaregistruje světlo, anténa se nasměruje přesně k jeho zdroji. Zastaví se, když jsou osvětlena obě čidla. Pokud se snažíme se zdrojem uhnout (ale stále svítíme na radar), nasměrování antény sleduje pohyb. Po zhasnutí opět radar přejde k vyhledávání zdroje.

Můžeme nastavit konstanty „min" a „max" podle použitého serva tak, aby radarová anténa měla větší úhel pohybu, servo ale nesmí nikdy narážet na mechanické dorazy, velmi brzy by se zničilo. Přednastavené hodnoty odpovídají normalizovanému impulzu pro servo (1,00 až 2,00 ms).

Trochu práce dá nalézt vhodnou polohu (úhel) obou čidel. Pokud anténa odskakuje od světla, prohodíme čidla. Jestliže nemíří přímo ke zdroji a při pohybu světla ho sleduje jen občasným skokem, rozevřeme čidla od sebe. Naopak když kolem nasměrování zakmitává, sevřeme je víc k sobě.

```
Program radar1;
{pr_037.pas}
{radar vyhledavajici svetlo}
uses ovl,crt,comint;
const min=115;
 max=230;
var poloha:word;
 smer:boolean;
 i:integer;

procedure pulz(delka:word);
 begin
  port[com1+3]:=128;
  port[com1+0]:=lo(delka);
  port[com1+1]:=hi(delka);
  port[com1+3]:=3;
  port[com1]:=$FF;
  delay(20);
 end;

begin
 init; clrscr;
 poloha:=round((max+min)/2);
 smer:=true;
 {pocatecni nastaveni do stredu}
 for i:=1 to 50 do pulz(poloha);
 repeat
  {nevidi svetlo - vyhledava}
  if (vstup1 and vstup2) then begin
   if smer then inc(poloha)
    else dec(poloha);
   if poloha<=min then smer:=true;
   if poloha>=max then smer:=false;
  end;
  {svetlo je vlevo}
  if (vstup1 and not(vstup2)) then
   inc(poloha);
  {svetlo je vpravo}
  if (not(vstup1) and vstup2) then
   dec(poloha);
  gotoxy(2,2);
  write('Stav : ',
   vstup1:10,vstup2:10);
  {kontrola a pohyb}
  if poloha<min then poloha:=min;
  if poloha>max then poloha:=max;
  pulz(poloha);
 until keypressed;
 if readkey=#0 then readkey;
end.
```

Radar lze stejně dobře postavit i z Lega, dokonce vypadá na pohled lépe. K pohonu v tom případě použijeme miniaturní servo nebo mikroservo.

Rozšíření úlohy vyžaduje postavit nový model, tentokrát s anténou pohyblivou kolem dvou os. Protože taková anténa je už podstatně těžší, nemůžeme stavět přímo na kotouči serva. Serva slouží jen k pohonu samonosné konstrukce.

Do antény umístíme čtyři optická čidla odchýlená od osy (vstupy 1, 2, 3 a 4), serva jsou připojena na oba porty (COM1 a COM2). Protože nemáme k dispozici čtyři univerzální posilovače, musí být alespoň jedno servo řízeno zapojením z obr. 31.

Činnost je naprosto stejná. Radar vyhledává a pak sleduje zdroj světla, tentokrát však ve dvou osách. Můžeme naprogramovat různé metody vyhledávání - postupně projíždět rozsah (jemně linku po lince), náhodné pohyby, hledání v rastru hrubé sítě a po zjištění zdroje přesné domíření, hledání v jedné ose s druhou ve středu, kdy se po nalezení zdroje zamíří druhá osa atd.

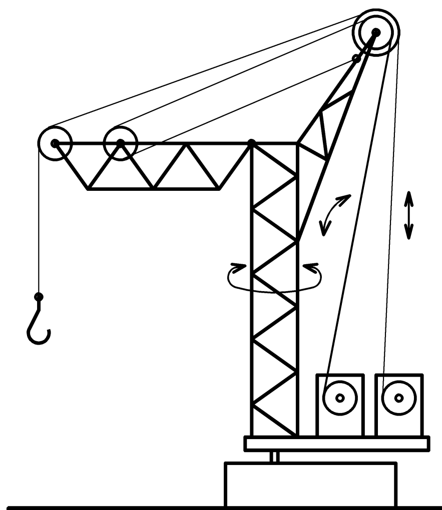
Je velmi zajímavé sledovat rychlost a spolehlivost, s jakou je ten který algoritmus vyhledávání schopen najít a zaměřit zdroj světla.

Umístíme-li radar tak, aby světlo (třeba od okna) přicházeno k němu zezadu a seřídíme vyšší citlivost všech čidel, je radar schopen pracovat i s odraženým světlem. Na vzdálenost jednoho až dvou metrů může vyhledávat i kousek bílého papíru, který držíme v ruce. Podmínkou však samozřejmě je, že místnost je tmavší a oblečení člověka také. Stejně tak se radar dobře „chytne" na bílou košili až ze čtyř metrů. Velmi efektní je vzít třeba do ruky větší bílou hračku (nejlépe letadlo) a přistávat s ním na „letišti", radar let přesně sleduje.

Jeřáb

Velmi široké možnosti možnosti řešit různé úlohy od nejjednodušších až po velmi náročné poskytují jeřáby všeho druhu. Využíváme nejméně dvě ovládané funkce (posun a zdvih) u portálových jeřábů, častěji však tři a více. K pohonu poslouží upravená serva s cívkami (lanovými bubny) nalepenými na kotouče - osvědčily se plastové cívky na spodní nit do šicího stroje.

Základní ovládání můžeme řešit jednoduše spínáním motorů čtyřmi nebo šesti klávesami. Na cívky se navíjí lanko (nejlépe tenké polyamidové, které je velmi ohebné a pevné záro-



Obr. 34. Složitější jeřáb

veň), konstrukce je ovládána přes kladky a kladkostroje, které současně zajistí i potřebné převody a vysokou sílu. Nadstavba ovládání pak umožňuje dopravit břemeno přesně na požadovanou polohu danou v pravoúhlých nebo válcových souřadnicích.

Příklad mechanického řešení trochu komplikovanějšího stavebního jeřábu je na obálce a na obr. 34. Ovládá se celkové otáčení, zdvih ramene a lano s hákem, vše samozřejmě obousměrně. Komplikace je v tom, že zdvih ramene není lineární funkcí otáčení příslušného servopohonu a navíc ovlivňuje i výšku háku nad podložkou.

Potřebujeme tři výstupy (výstup 1, 2 a 3) a tři relé k reverzaci chodu, jedno tedy musíme doplnit na výstup 4. Jednoduchý program, kterým lze jeřáb ručně z klávesnice ovládat, reaguje na klávesy kurzoru, PgUp a PgDn. Pro jednoduchost dovoluje pohyb vždy pouze jedné funkce současně, ne pohyby kombinované.

```
Program jerab;
{pr_038.pas}
{rucni ovladani jerabu}
uses ovl,crt;
var b1,b2:char;

begin
 init; clrscr;
 writeln(' Rucni ovladani jerabu');
 writeln;
 writeln(' PgUp ... rameno nahoru');
 writeln(' PgDn ... rameno dolu');
 writeln(' ^    ... bremeno nahoru');
 writeln(' v    ... bremeno dolu');
 writeln(' =>   ... otaceni vpravo');
 writeln(' <=   ... otaceni vlevo');
 writeln(' SPC  ... konec programu');
 writeln;
 repeat
  repeat until keypressed;
  b1:=readkey;
  if b1=' ' then begin
   init;
   halt;
   end;
  if b1=#0 then begin
   b2:=readkey;
     case b2 of
      #73: begin
            zapni_re1;
            zapni1;
            delay(50);
            vypni1;
           end;
      #81: begin
            vypni_re1;
            zapni1;
            delay(50);
            vypni1;
           end;
      #72: begin
            zapni_re2;
            zapni2;
            delay(50);
            vypni2;
           end;
      #80: begin
            vypni_re2;
            zapni2;
            delay(50);
            vypni2;
           end;
      #75: begin
            zapni4;
            zapni3;
            delay(50);
            vypni3;
           end;
      #77: begin
            vypni4;
            zapni3;
            delay(50);
            vypni3;
           end;
     else pip
     end;
    end;
   until false;
  end.
```

Dalším stupněm ve zvládnutí jeřábu je automatická kompenzace výšky háku při spouštění nebo zdvihání ramene. Začne být výhodné mít možnost řídit pohyby jeřábu nejen co do zapnutí a vypnutí, ale i rychlost. Stále si vystačíme s možnostmi přípravku. Otáčení ponecháme jako funkci spínanou, rameno a lano budeme řídit výstupy TxD portů COM1 a COM2. To nám umožní současný koordinovaný pohyb dvou pohonů, jehož cílem je zachovat výšku háku nad podložkou při přenášení břemene od paty jeřábu na okraj jeho dosahu.

Můžeme zkusit doplnit funkci, která najede s hákem automaticky na dané místo zadané souřadnicemi. Zatím budeme předpokládat, že prostor kolem jeřábu je zcela volný. Jakmile začneme uvažovat o práci v souřadnicích, musíme doplnit snímání a sledování polohy všech servopohonů.

V nejjednodušším případě doplníme ke každému pohonu jedno optické čidlo, které vytvoří impulz řekněme na každých 5 mm odvinutého lana. Na této délce pochopitelně nezáleží, ale je nutné ji změřit a počítat s ní. Délku pohybu pak odečítáme jako počet impulzů, směr pohybu předpokládáme podle zadaného směru (výstup na relé). Na počátku musíme ručně najet do výchozího bodu. Tento způsob ovládání není příliš přesný, občas nějaký ten impulz „uteče" a kalibraci ve výchozím bodě je třeba často opakovat. Také pružnost a vůle celé konstrukce - uvažujeme stále stavbu z Merkuru - vykonají na nepřesnostech své.

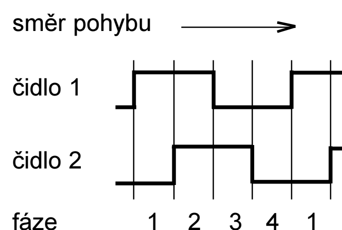
Přidáním dalších čidel tak, aby u každého pohonu byla dvě s fázově posunutým snímáním (obr. 35), dovolí sledovat velikost i smysl skutečného pohybu na střídání fází mnohem přesněji. Stále bez problémů vystačíme s přípravkem Interface PC, šest vstupů na něm máme.

Dostáváme se k úlohám, u kterých se již nebude měnit model, ale jen velmi poroste složitost jeho řízení. Je dobré, když je model co nejtužší a pohybuje se bez vůlí, pokud musíme použít řemínky, pak pouze kvalitní tuhé s malou pružností. Dostanou se jako náhradní díly ke starým typům magnetofonů. Bude záležet nejen na přesnosti, ale i na dynamických vlastnostech jeřábu.

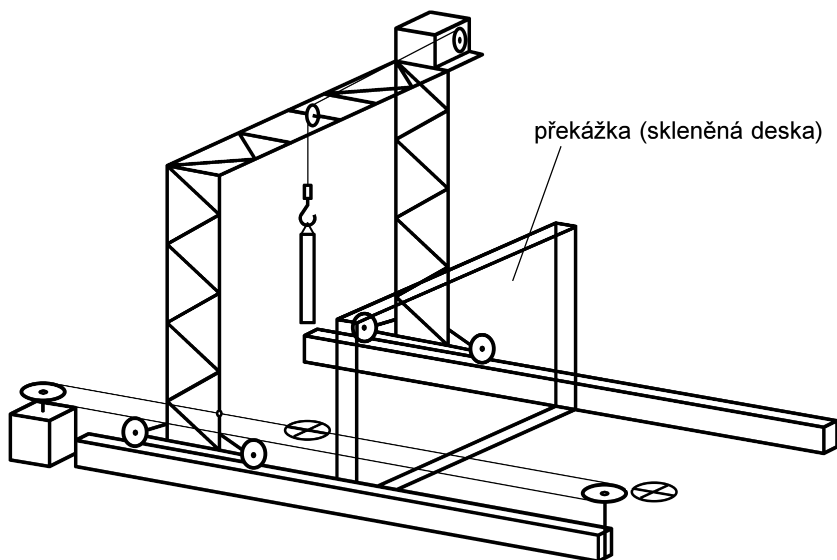
Jako první z obtížnějších úloh můžeme zkusit automaticky řídit jeřáb z aktuální polohy do souřadnicemi zadaného bodu, ovšem v reálném prostředí s překážkami. Nepracujeme již tedy ve volném prostoru, ale vedle jeřábu postavíme „domy" (třeba polystyrénové kostky) o různé výšce. Některé mohou i bránit v otáčení jeřábu, pokud není rameno zvednuto. Budeme přenášet břemena ze „země" nebo horní plochy „domu" na „zem" nebo horní plochu jiného „domu".

Do programu musíme zadat mapu překážek (včetně výšek, tedy třírozměrnou) v souladu s modelem. Program musí nést břemeno tak, aby nenarazil částí jeřábu ani břemenem do překážky. Svou roli samozřejmě hraje velikost a tvar břemene, může to být i svisle zavěšená tyč. Nezáleží na rychlosti pohybu, jen na přesnosti. Stačí ovládat vždy jen jednu funkci v daném okamžiku, snaha o rychlost se vymstí. Pracujeme naopak velmi pomalu, abychom nemuseli uvažovat houpání břemene na laně. Pokusná břemena volíme lépe větší a lehčí, třeba kostky vyříznuté ze stavebního polystyrénu (nedělá kuličky a nešpiní okolí), protože se po rozkývání odporem vzduchu rychle ustálí.

Následují úlohy, při nichž budeme už využívat popisu dynamických vlast-



Obr. 35. Signály z čidel s fázově posunutým snímáním



Obr. 36. Portálový jeřáb s překážkou

ností jeřábu i břemene. V těchto případech budeme naopak používat jako břemena menší předměty s větší hmotností, např. velký železný šroub o hmotnosti kolem 100 g. Máme předmět zavěšený na háku na zemi těsně u paty jeřábu. Naším úkolem je ho co nejrychleji přenést asi do 70 % vzdálenosti od paty (100 % = dosah jeřábu).

Jakmile budeme pracovat na rychlost, břemeno na laně se při pohybu nutně rozhoupe. Podstata úlohy je v tom, jak co nejrychleji stabilizovat polohu břemene pohybem ramene a přitahováním či povolováním lana, aby při spuštění na podložku bylo přesně v daném místě a skoro v klidu. Otáčení jeřábu využívat nebudeme, vše se děje v rovině bez trojrozměrných překážek.

Pokud by někomu předchozí úloha ještě nestačila, může doplnit otáčení jeřábu. Zatímco dosud se břemeno chovalo jako slušně vychované kyvadlo, nyní na něj bude působit i odstředivá síla při otáčení a bude opisovat podstatně složitější obrazce. Stabilizace bude tedy muset také využívat všechny tři pohony. Vyřešit tuto úlohu při automatickém řízení, kdy nemáme informaci o skutečné aktuální poloze břemene, je velmi obtížně. Dokonalého výsledku nedosáhneme nikdy, protože nikdy nebude souhlasit matematický model úlohy s reálným modelem. Jde jen o to, aby program ztlumil kývání co nejlépe (resp. alespoň znatelně).

Pokud si chcete vyzkoušet schopnosti jeřábníka v praxi, stačí nechat automat co nejrychleji donést břemeno nad cíl (jako by viselo na tuhé svislé tyči) a předat řízení na ruční ovládání tlačítky. Dál se snažte uklidnit kývání ručně. Je to mnohem horší, než lovit plyšové medvídky z automatu na pouti ...

Ani to však není vrchol možností. Poslední úloha může vypadat třeba takto. Máme portálový jeřáb, ovládané funkce jsou pojezd mostu a navíjení lana s hákem. Ovládání vozíku nemáme, vše se děje v jedné rovině. Je důležité, aby pohony umožňovaly svižný pohyb a model měl větší rozměry - výšku třeba kolem metru. Na háku je zavěšeno těžké břemeno (50 až 100 g) ve tvaru svislé tyče (ocelový drát 40 cm dlouhý). Máme ho přenést přes překážku a stabilizovat na místě určení. Překážka stojí kolmo na dráhu pojezdu mostu a je tak vysoká, že ani když zdvihneme tyč maximálně nahoru, neprojde nad překážkou (jen o kousek, třeba o 20 mm). Situace je na obr. 36.

Jedinou cestou, jak dostat břemeno na druhou stranu překážky, je pohybem mostu ho řízeně rozhoupat, pak využít okamžiku, kdy je konec nad úrovní překážky, „přehodit" ho na druhou stranu a tam co nejrychleji stabilizovat polohu. Břemeno se nesmí ani dotknout překážky, jinak je pokus neúspěšný. Opět můžeme zkusit zvládnout tuto úlohu ručním řízením, je to svým způsobem mnohem snazší než vytvoření matematického popisu a následné naprogramování. Pro pokusy použijeme překážku z tuhého papíru, třeba čtvrtky.

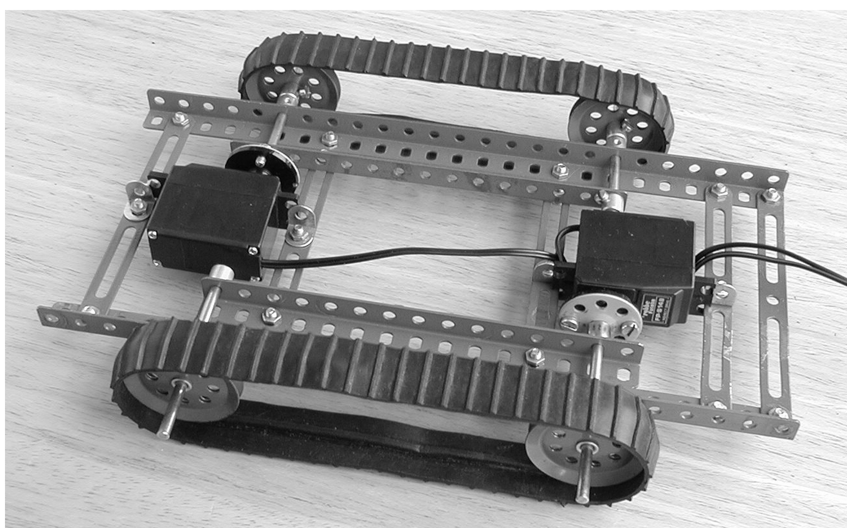
Mechanicky, ručním řízením i programově je tato úloha plně řešitelná a její obtížnost dost závisí na tom, jaký přesah bude mít zavěšená tyč a překážka. Tato úloha na trochu větším modelu se řešila i v rámci laboratorních cvičení na vysoké škole, jen jako překážka sloužila menší tabulka z tenkého skla. Kritérium úspěšnosti bylo jednoduché - nerozbít sklo.

Naznačené úlohy, které je možno řešit s modely jeřábů, mají dokumentovat obrovské rozpětí možné náročnosti. Program na jednoduché ruční ovládání může napsat desetileté dítě (které mimochodem zvládne ručním řízením po jistém zácviku i ty nejnáročnější uvedené případy) se znalostmi matematiky odpovídajícími věku. Při programování složitějších úloh musí zabrat středoškoláci (vyhýbání se 3D překážkám, uklidnění jednoduchých kyvů v rovině) a nejnáročnější úlohy zatopí snadno i vysokoškolákům. Časová náročnost je od asi půl hodiny po několik desítek hodin (jen na popis a program, model uvažujeme hotový a připojený).

Současně se také dostáváme na mez, kterou klade rychlost PC. S několikrát uvedeným počítačem 386SX20 jsou nejnáročnější úlohy již problematické, protože potřebujeme přesnější řízení v čase a rychlejší řešení diferenciálních rovnic matematického popisu. Počítač Pentium 75 MHz, který je z hlediska bazaru také již neprodejným šrotem, to však zvládá dobře.

Pásové vozidlo

Řízení různých vozítek počítačem vždy naráží na problém připoutání vozítka svazkem kabelů k PC. Ani dálkové bezdrátové ovládání pomocí sériově vysílaných povelů malým modulem pracujícím na 433 MHz sice není nic složitého, ale pro první pokusy postačí kabel. Zvolíme takové vozidlo, které je „od přírody" silné, pomalé a jeho řízení přináší něco nového. Bude to robot na pásovém podvozku.



Obr. 37. Podvozek pásového vozidla

Mechanické řešení vychází opět z Merkuru, který pryžové pásy i další potřebné díly v některých sadách obsahoval. Blíže napoví obrázek holého podvozku s pohony (obr. 37). Pohon zajišťují dvě upravená modelářská serva připojená přes relé (pro změnu směru) na výstupy 1 a 2. S kabely dlouhými 2 až 3 metry lze projíždět udanou dráhu, překonávat překážky nebo projíždět na přesnost brankami.

Úlohou je připravit zjednodušené ruční řízení pohonu klávesami „1” až „5” tak, že na povel „1” točí podvozek naplno vlevo (levý pás naplno vzad, pravý naplno vpřed), „2” znamená ostře vlevo (levý pás stojí, pravý naplno vpřed), „3” rovně vpřed. Klávesy „4” a „5” řídí analogicky pravou zatáčku. „0” znamená rovné couvání, mezerník program ukončuje.

```
Program pasak;
{pr_039.pas}
{rucni ovladani pasoveho vozidla}
uses ovl,crt;
const krok=100;
var b:char;

begin
 init; clrscr;
 writeln(' Pasove vozidlo');
 writeln;
 writeln(' 1   ... ostre vlevo');
 writeln(' 2   ... vlevo');
 writeln(' 3   ... rovne vpred');
 writeln(' 4   ... vpravo');
 writeln(' 5   ... ostre vpravo');
 writeln(' 0   ... vzad');
 writeln(' SPC ... konec programu');
 writeln;
 repeat
  repeat until keypressed;
  b:=readkey;
  case b of
   ' ': begin init; halt; end;
   '1': begin
         vypni_re1;
         zapni_re2;
         zapni1; zapni2;
         delay(krok);
         vypni1; vypni2;
        end;
   '2': begin
         vypni_re1;
         zapni1;
         delay(krok);
         vypni1;
        end;
   '3': begin
         vypni_re1;
         vypni_re2;
         zapni1; zapni2;
         delay(krok);
         vypni1; vypni2;
        end;
   '4': begin
         vypni_re2;
         zapni2;
         delay(krok);
         vypni2;
        end;
   '5': begin
         zapni_re1;
         vypni_re2;
         zapni1; zapni2;
         delay(krok);
         vypni1; vypni2;
        end;
   '0': begin
         zapni_re1;
         zapni_re2;
         zapni1; zapni2;
         delay(krok);
         vypni1; vypni2;
        end;
   else pip;
   end;
 until false;
end.
```

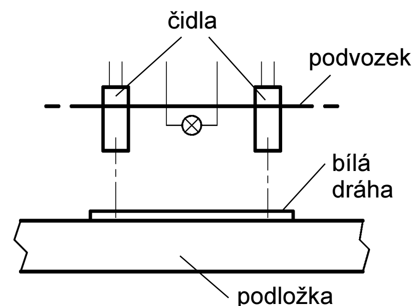
Tato úloha také nevyžaduje zvláštní znalosti. Dá se následně rozšířit na automatické řízení vozítka po určité dráze (čtverci, kruhu nebo rovně vpřed, otočit a zpět), jemné řízení směru s využitím ovládání rychlosti obou pásů apod. Má sloužit především jako příprava pro úlohu následující. Přesto většinou i samostatně silně zaujme, zejména pokud je podvozek kryt atraktivní kapotou.

Robot na lince

Programátorsky zajímavější, atraktivní jako hračka a ne moc složité je doplnění pásového podvozku o automatické sledování vyznačené dráhy. Na tmavou podložku (sololitovou desku) nakreslíme bílou barvou dráhu širokou asi 2 až 3 cm. Můžeme použít i kus tmavého jednobarevného koberce a na něj nalepit proužky bílé látky (stuhy) nebo papíru. Dráha nemusí být rovná, může obsahovat i zatáčky (poloměr asi 15 cm), výhybky (rozdvojení a sloučení čar) a ostrá zakončení.

Na podvozek umístíme žárovku a dvě optická čidla podle obr. 38 a připojíme je na vstupy 1 a 2. Čidla seřídíme tak, aby viděla odražené světlo od bílé linky, ale nereagovala na tmavý podklad. Pokud jsou osvětlena obě, pojede robot rovně. Jakmile jedno z čidel vybočí z dráhy, začne podvozek zatáčet tak, aby odchylku vyrovnal - sleduje i klikatou čáru. Pokud obě čidla sjedou z dráhy, začne se robot otáčet na místě vlevo, dokud nezachytí opět linii (je dobré případně omezit dobu otáčení asi na jednu otáčku, jinak na sebe namotá celý kabel).

Aby bylo možné využít i výhybky a dokonce i křižovatky, má program ruční řízení, ale proti předchozímu příkladu jen zjednodušené - vlevo, vpravo,



Obr. 38. Umístění žárovky a dvou optických čidel na podvozku pásového vozidla

vpřed, vzad. Pokud předáme řízení automatu, jede po lince. Na výhybce pokračuje sám (většinou) rovně, pokud chceme zatočit, dáme krátce povel ke změně směru (obr. 39). Jestliže na rovné trati vybočíme ručně vlevo, robot otočí a vydá se zpět po stejné dráze. Vybočíme-li vpravo, robot se vrátí na původní trať. Program má testovací režim čidel, aby se usnadnilo jejich seřízení.

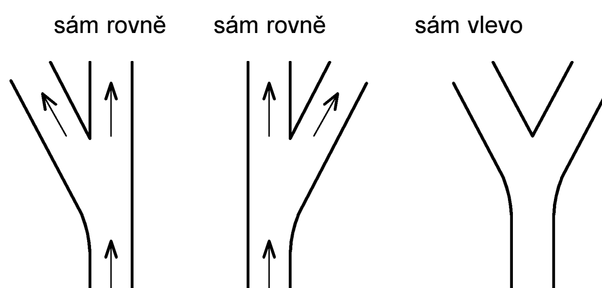
```
Program linka;
{pr_040.pas}
{pasove vozidlo na lince}
uses ovl,crt;
const krok=100;
var b1,b2:char;

procedure vlevo;
 begin
  vypni_re1;
  zapni1;
  delay(krok);
  vypni1;
 end;

procedure vlevo2;
 begin
  vypni_re1; zapni_re2;
  zapni1; zapni2;
  delay(krok);
  vypni1; vypni2;
 end;

procedure vpravo;
 begin
  vypni_re2;
  zapni2;
  delay(krok);
  vypni2;
 end;

procedure vpravo2;
 begin
  vypni_re2; zapni_re1;
```



Obr. 39. Výhybky na vyznačené dráze pro pásový podvozek

```
   zapni2; zapni1;
   delay(krok);
   vypni2; vypni1;
  end;

 procedure vpred;
  begin
   vypni_re1;
   vypni_re2;
   zapni1; zapni2;
   delay(krok);
   vypni1; vypni2;
  end;

 procedure vzad;
  begin
   zapni_re1;
   zapni_re2;
   zapni1; zapni2;
   delay(krok);
   vypni1; vypni2;
  end;

 begin
  init; clrscr;
  writeln(' Pasove vozidlo na lince');
  writeln;
  writeln(' Rucni rizeni kurzory');
  writeln(' A   ... automat');
  writeln(' T   ... test cidel');
  writeln;
  writeln(' SPC ... konec programu');
  writeln;
  repeat
   repeat until keypressed;
   b1:=readkey;
   case b1 of
    ' ': begin init; halt; end;
    't','T': repeat
          gotoxy(2,9);
          writeln(vstup1:10,vstup2:10);
          delay(100);
          gotoxy(2,9);
          writeln('  ');
          until keypressed;
    'a','A': repeat
          if (vstup1 and vstup2)
           then vlevo2;
          if (not vstup1 and not
           vstup2) then vpred;
          if (vstup1 and not
           vstup2) then vlevo;
          if (not vstup1 and
           vstup2) then vpravo;
          until keypressed;
    #0: begin
        b2:=readkey;
        case b2 of
         #75: vlevo2;
         #72: vpred;
         #77: vpravo2;
         #80: vzad;
         else pip;
         end;
        end;
   else pip;
   end;
  until false;
 end.
```

Jak řízení uvedeným programem funguje si můžete prohlédnout na videu, které je umístěno spolu s programy na internetových stránkách časopisu Pratická elektronika A Radio na adrese *www.aradio.cz* (71 sekund záznamu, délka 1,3 MB).

Dalších variací na toto téma je bezpočet. Můžeme doplnit dopředu tykadlo s mikrospínačem, kterým robot „hmatá" překážky před sebou, a zkoušet různé algoritmy pro objíždění překážek. Lze instalovat nahoru fotočidlo a naprogramovat, aby robot na světelný povel otočil a vydal se zpět. Dvě světelná čidla na „radaru" umožní vyhledat zdroj světla, opustit dráhu a vydat se k němu. Pokud světlo zmizí, hledat podle určitého algoritmu bílou dráhu nebo rovnou pamatovat ujetou vzdálenost a tvar cesty za světlem a vydat se zpět „po vlastních stopách".

Výtah

Z Merkuru si můžeme postavit model výtahu v několikapatrovém domě a simulovat jeho obsluhu. Základ konstrukce tvoří věž z Merkuru nebo čtyř duralových úhelníků. K pohonu výtahu slouží opět upravené servo s navíjecí cívkou na lanko, přišroubovanou na kotouči (obr. 40). Servo je připojeno na výstup z prvního D/A převodníku a univerzální posilovač. Na stejném hřídeli jako cívka je i další kolečko s osmi otvory pro šrouby, které využijeme jako clonku pro fotočidlo (vstup 1). Fotočidlo snímá pohyb výtahu.

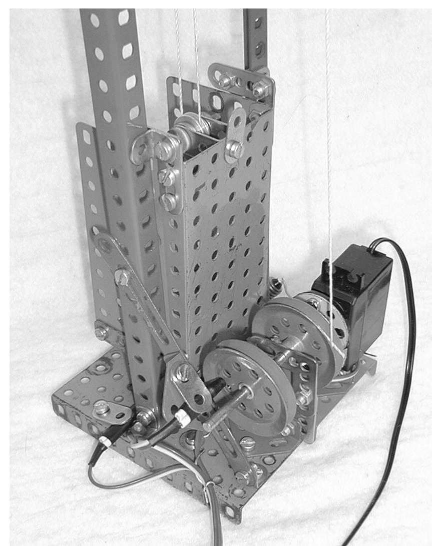
V pravidelných odstupech přišroubujeme čtyři plošiny znázorňující jednotlivá podlaží. Jako ve skutečném výtahu může kabina sjet ještě asi o půl patra níž, než je nejspodnější podlaží, tam je optické čidlo detekující kabinu (vstup 2) (doraz).

Na vstupy 3 až 6 připojíme tlačítka (mikrospínače). Šlo by samozřejmě ovládat výtah pouze z klávesnice PC, ale takhle to bude zajímavější, zejména pokud jednotlivé spínače umístíme na svislý panel vedle pater. Těmito tlačítky se bude výtah přivolávat. Volbu cíle jízdy (tlačítka v kabině výtahu) simuluje na klávesnici tlačítko „P" (přízemí) a „1" až „3".

Předem si změříme počet kroků potřebných k přejetí do jednotlivých pater (pokud se lanko navíjí na buben, průměr se postupně nepatrně mění, ale rozdíl není podstatný). Příslušnou výšku pater nastavíme v proměnné patro. Obslužný program musí v první řadě zjistit, kde je kabina. Proto jede dolů tak dlouho, až optické čidlo dorazu zachytí kabinu, tím se zorientuje. Přejede do přízemí a očekává povely cestujících.

Je důležité přejíždět s kabinou relativně pomalu, aby při zastavení na čidle pohybu „neutekly" impulzy. Protože kabina z dílů Merkuru je dost těžká a výrazně se projeví na rychlosti serva, jsou odlišeny rychlosti pro jízdu nahoru a dolů.

V nejjednodušší verzi (obvyklé u starších výtahů) čeká výtah na povel k přistavení do daného patra nebo povel z kabiny. Příkaz přijme a vykoná,



Obr. 40. Strojovna výtahu

přičemž další povely neakceptuje ani si je nepamatuje. Navíc program dovoluje kdykoli obnovit nastavení správné polohy sjetím do spodní polohy. Program se ukončuje mezerníkem.

```
Program vytah;
{pr_041.pas}
{rizeni kabinoveho vytahu}
uses ovl,crt;
var poloha,i:integer;
 patro:array [0..3] of integer;
 kl:char;

procedure jed(p:integer);
begin
 if poloha=patro[p] then exit;
 if poloha<patro[p] then zapni_re1
  else vypni_re1;
 delay(200);
 if poloha<patro[p] then
  rychlost_1(15) else rychlost_1(4);
 repeat
  repeat until not vstup1;
  repeat until vstup1;
  if poloha<patro[p] then
   inc(poloha) else dec(poloha);
 until poloha=patro[p];
 rychlost_1(0);
 vypni_re1;
 while keypressed do readkey;
 delay(500);
end;

procedure sjed;
begin
 vypni_re1;
 delay(200);
 rychlost_1(4);
 repeat until vstup2;
 rychlost_1(0);
 poloha:=0;
 delay(100);
 zapni_re1;
 delay(200);
 jed(0);
end;

begin
 patro[0]:=3;
```

```
  patro[1]:=20;
  patro[2]:=37;
  patro[3]:=54;
  init;
  sjed;
  repeat
   if vstup3 then jed(0);
   if vstup4 then jed(1);
   if vstup5 then jed(2);
   if vstup6 then jed(3);
   if keypressed then begin
    kl:=readkey;
    case kl of
     #0:  readkey;
     ' ': halt;
     'p','P': jed(0);
     '1': jed(1);
     '2': jed(2);
     '3': jed(3);
     'k','K': sjed;
     else pip;
     end;
    end;
  until false;
 end.
```

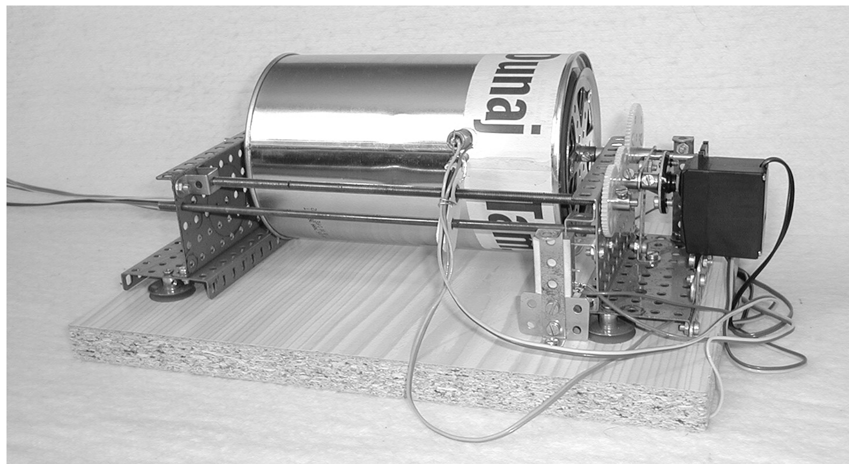
Stejně jako i jiné úlohy je možné oživený model výtahu dál rozvíjet, především co do inteligence ovládání. Výtah si může pamatovat pořadí požadavků a postupně je vykonávat, sbírat cestující postupně i podle požadavků došlých v průběhu jízdy nebo dokonce sledovat četnost nástupů a cílových stanic pasažérů. Učit se automaticky parkovat v patře, z něhož může přijít podle dosavadních zkušeností požadavek nejpravděpodobněji a tím minimalizovat dobu nutnou k přistavení výtahu. Můžeme pozorovat chování moderních výtahů a snažit se ho napodobit.

Scanner

Z dílů Merkuru je možné celkem snadno zkonstruovat i plotter nebo scanner. Ostatně před lety byl populární podobný zapisovač Alfi. Zkusíme si postavit jednoduchý bubnový scanner na formát A5, který je schopen nasnímat a zobrazit jednoduchou kresbu nebo titulek v novinách (obr. 41). Účelem není získat hezké a věrné obrázky z fotografií, ale demonstrovat na co nejjednodušším vybavení i programu funkčnost. Stavebnici Merkur budeme muset doplnit několika dalšími díly.

Základ tvoří kvůli požadované tuhosti obdélníková deska z dřevotřísky o rozměrech asi 20x30 cm. Na ploše jsou přišroubovány díly Merkuru, které nesou závěsy válce vyrobeného z čisté plechovky od technického ředidla. Stejně by šlo použít i třeba umytou plechovku od nějakého nápoje. Do čel plechovky jsou vruty přišroubována kola Merkuru.

Pohon scanneru zajišťuje jediné upravené modelářské servo přes převodovku z ozubených kol (díl Merkuru). Servo otáčí jednak bubnem (na něj izolepou přichytíme snímanou předlohu), jednak dlouhým šroubem, který na matce unáší jezdec se zdrojem světla a snímačem připojeným na vstup 1. Jezdec je vyroben z kousku plastické hmoty a na matce drží částečně namáčknutím, částečně přilepením sekundovým lepidlem. Čidlo je na něj přivázáno drátem nebo přilepeno izolepou.



Obr. 41. Bubnový scanner

Otvory v ozubeném kole na hřídeli bubnu jsou zaslepeny černou páskou, jen jeden je průchozí a propouští světlo žárovky na druhé fotočidlo (vstup 2). Tím je určen počátek snímání během otáčky.

V krajní poloze narazí jezdec do tlačítka mikrospínače (vstup 3) a sepne ho, čímž dá počítači jednoznačnou informaci o své poloze. Jezdec nese zdroj světla a fotočidlo připojené na vstup 1. Scanner v této verzi nerozlišuje stupně jasu, jen černou a bílou. Obě optická čidla jsou naprosto stejná jako ve všech předchozích úlohách.

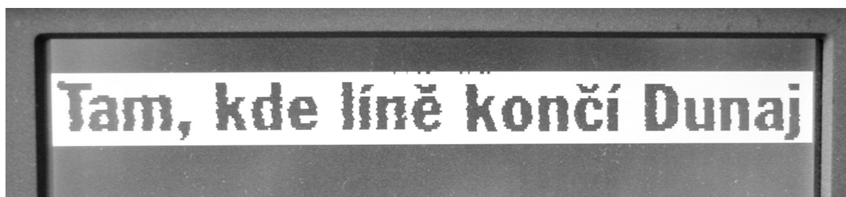
Obslužný program vykonává jen nejnutnější činnosti. Na začátku se servo otáčí tak dlouho, až jezdec narazí do mikrospínače. Pak se smysl otáčení změní a čidlo linku po lince snímá předlohu. Současně se na šroubu posunuje.

Snímání bodů v průběhu jedné otáčky není nijak synchronizováno s pohybem bubnu, pouze čidlo 2 detekuje počátek snímání a potom je vše již jen časováno programem. Jakákoli nerovnoměrnost chodu pohonu se tedy projeví přímo na kresbě. Přesto jsou výsledky slušně použitelné, jak se můžete přesvědčit z fotografie (obr. 42).

Jako příklad byl snímán titulek vystřižený z novin. Novinový papír a tisk je výhodný pro pokusy, protože je kontrastní a má matný povrch - neházi odlesky. Jak se tato předloha sejmula a zobrazila na LCD displeji starého notebooku, posuďte sami (viz obr. 42). Není to sice dokonalé, ale čitelné určitě ano.

```
Program scanner;
{pr_042.pas}
{jednoduchy scanner A5}
uses ovl,crt,graph;
var gd,gm,i,j,k:integer;

begin
 init; clrscr;
 {repeat writeln(vstup1:10)
  until keypressed; halt;}
 writeln;
 writeln(' Presun na pocatek');
 zapni_re1;
 zapni1;
 repeat until vstup3;
 vypni1;
 vypni_re1;
 gd:=vga;
 gm:=vgahi;
 initgraph(gd,gm,'');
 i:=1; j:=1;
 zapni1;
 repeat until not vstup2;
 repeat until vstup2;
 repeat
  repeat
   repeat until vstup2;
   if vstup1 then begin
    putpixel(638-j,i+30,0);
    putpixel(638-j,i+31,0);
    end
    else begin
    putpixel(638-j,i+30,15);
    putpixel(638-j,i+31,15);
    end;
```



Obr. 42. Text, přečtený bubnovým scannerem, zobrazený na LCD displeji notebooku

```
      inc(j);
      for k:=1 to 20 do;
      delay(3);
    until ((j>638) or (not vstup2));
    i:=i+2;
    j:=1;
    repeat until not vstup2;
  until i>180;
  vypni1;
  repeat until keypressed;
  if readkey=#0 then readkey;
  closegraph;
end.
```

Tento jednoduchý program není možné převzít bez úprav souvisejících s konstrukcí scanneru, funkce programu závisí na průměru bubnu, převodech, rychlosti PC atd.

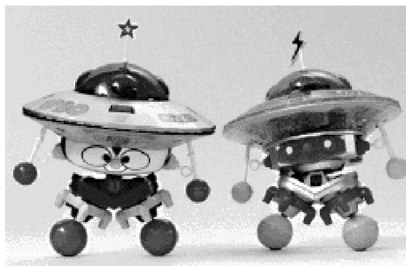
Jednoduchá mechanika scanneru snese samozřejmě další vylepšení, zejména zmenšení fotonkou snímaného bodu a zavedení synchronizace čtení v průběhu otáčky. Velmi zajímavé je i zařadit za snímací prvek A/D převodník, úplně stačí čtyřbitový, pomocí něhož rozlišíme stupně jasu. Potom se dá začít experimentovat i se snímáním fotografií.

Měření

Ve všech úlohách jsme vystačili víceméně bez obvyklých převodníků A/D. Jejich zvládnutí otevírá další široký prostor pro aplikace a pokusy. Většina často publikovaných návodů se však soustředí především na měření jako taková (teploty, průběhu napětí, charakteristik tranzistorů atd.), která jsou pro menší děti nezajímavá, protože nerozumí jejich smyslu.

Řada úloh využívajících jednoduché a levné (avšak nikoli příliš přesné) prostředky z oblasti měření je výborně zpracována v [3]. Najdeme tu nejen napěťové převodníky D/A a A/D, již zmíněné snímání charakteristik tranzistorů, ale třeba i jednoduchý osciloskop, realizovaný pomocí běžné zvukové karty nebo měření délek z obrazové informace pomocí videokamery.

Záleží jen na konkrétních podmínkách, zda tyto úlohy jsou dobře přijímány nebo ne. Vždy je třeba najít především nějaké pro děti smysluplné využití, což může být něco zcela odlišného od chápání smysluplnosti námi, dospělými. Např. technicky elegantní a nepříliš náročné vybudování meteorologické ústředny má jen nepatrnou naději na úspěch vedle miniaturní funkční televize do pokojíčku pro panenku Barbie (bodový displej LCD ze starého Tamagotchi připojený k PC, které na něm zobrazuje sled obrázků, pokud si někdo chce pohrát, tak i animací). Přestože druhý příklad je spojen s mnohem větší, piplavou a doslova „hnusnou“ prací jak z hlediska přípravy techniky tak programu, pro ty, kdo vydrží, je úspěch jistý.



Obr. 43. Dálkově ovládaná figurka

Roboti a hříčky v zahraničí

Stavba jednoduchých funkčních modelů čehokoli a jejich řízení počítačem je v zahraničí ne moc rozšířená, ale o to více propracovaná zábava. Přitom zhusta nejde o činnost zaměřenou na děti a výuku, a už vůbec ne na výuku programování, převažuje kategorie pánů středního až pokročilého věku, kteří toto provozují jako své hobby. U nás, přestože slovo robot je, jak známo, českého původu, podobné aktivity téměř neexistují.

Na internetu můžeme najít řadu velmi zajímavých stránek, na nichž se prezentuje mnoho amatérských tvůrců i firem z této oblasti. Jednou z nich je např. *www.robotstore.com* , kde najdeme širokou komerční nabídku hotových hříček, stavebnic i dílů. Na obr. 43 je dálkově ovládaná figurka, která díky vnitřnímu gyroskopu dokáže skutečně stát na dvou nohách (navíc dole zakončených kuličkami) a trochu se pohybovat. Jak náročná je to úloha, posoudí především ten, kdo to zkusil.

Robotstore je především místem, kde se soustřeďuje zboží mnoha nejrůznějších výrobců. Jsou tu roboti autonomní, programovatelní, ovládaní počítačem, specializované stavebnice LEGO, mechanické ruce, senzory, moduly syntézy řeči, kybernetičtí mazlíčci (jako známý Aibo), literatura, video i plakáty s danou tématikou.

Další stránky s podobným zaměřením, které se vyplatí navštívit, jsou *www.hobbyrobot.com* (obr. 44). Některé konstrukce se přitom už trochu vymykají vžité představě o modelech a hračkách. Třeba v sortimentu podvozků pro roboty najdeme takové, které se vejdou do dlaně i ty, které velikostí připomínají ze všeho nejvíc servírovací stolek.

Všeobecné oblibě se těší především neobvyklé tvary a provedení připomínající kosmické lodě, létající talíře nebo vícenohé kráčející stroje. Řadu jich najdeme na stránkách *www.lynxmotion.com* nebo *www.mrrobot.com* , nejmenší počet noh bývá čtyři, obvyklý šest až osm (obr. 45). Strojová chůze je velmi atraktivní na pohled.

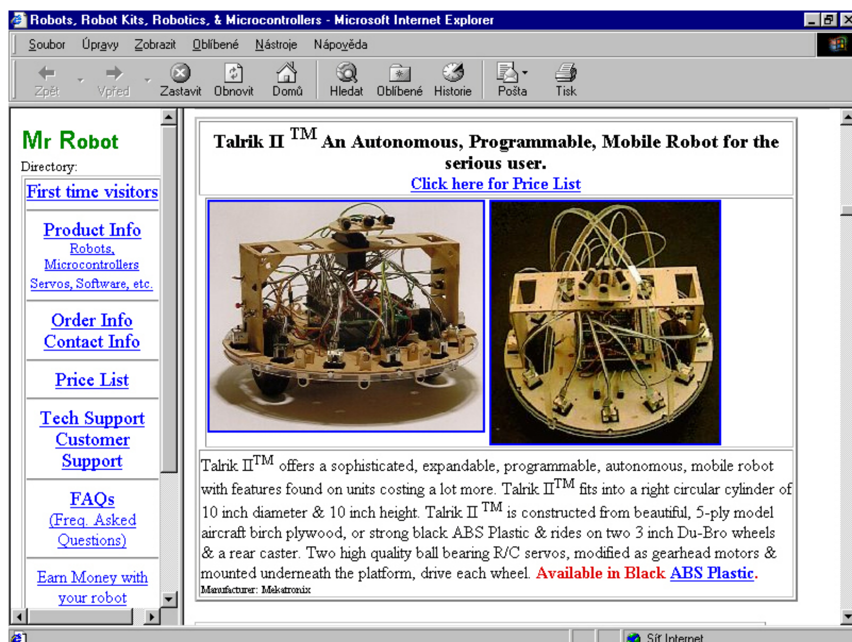
Uvedené konstrukce robotů se zaměřují zejména na mechanickou stavbu a kompletování z koupených dílů, méně na programování, a pokud už ano, pak většinou na programování jednočipových počítačů, protože jde téměř výhradně o autonomní stroje. Řízení prostřednictvím PC je jen výjimečné.

Ani velké firmy zaměřené na technické stavebnice nenechaly oblast robotů bez povšimnutí. Již několikrát zmíněné LEGO a výtvory z něj lze oživit třeba pomocí jednotky RCX 1.0, které byl věnován samostatný článek v časopise Elektor ([8]). Jednotka se skládá z mikropočítače a připojených čidel a akčních členů (pohonů), vše je samozřejmě vestavěno do plastových dílů tak, aby je bylo možné použít zcela přirozeně při stavbě ze stavebnice. Obslužný program pod Windows vyžaduje PC Pentium 166 MHz a 16 MB RAM, tedy nijak vysoké nároky nemá. Podrobnosti najdete na stránkách *www.legomindstorms.com* .

Celá sada, která se v současnosti v novější verzi prodává pod označením Lego Mindstorms Robotics Inven-



Obr. 44. Stránky www.hobbyrobot.com



Obr. 45. Kráčející stroje na stránkách www

tion System 2.0 obsahuje více než 700 dílů stavebnice, CD s programem pro PC, infračervené dálkové ovládání připojitelné pře USB port k PC, dva pohony, dva dotekové senzory, světelné čidlo a další díly (obr. 46).

Stavebnice je jistě velmi dobře propracovaná, jen její cena kolem 200$ je na naše poměry dost vysoká. Prodávají se i samostatné díly, jak už to ale tak bývá, jejich cena je z našeho pohledu ještě méně přijatelná. Např. jednoduchá kostka s optickým čidlem přijde na 30$, obdobně tlakové čidlo (mikrospínač v kostce Lego) kolem 18$.

Firmu, která se cíleně a již po mnoho let věnuje funkčním výukovým pomůckám, najdeme na stránkách *www.fischertechnik.com* . Pamatuji se, že jsem asi kolem roku 1992 velmi obdivoval rozsáhlé modely výrobních linek na jedné výstavě v Praze. Vše až neuvěřitelně fungovalo, dopravní pásy dopravovaly, manipulátory přenášely, obráběcí stroje byly plně v pohybu. To vše pod řízením několika tehdy špičkových PC 386SX.

Aby ještě více vyniklo určení těchto pomůcek, byly počítače obsluhovány dětmi ve věku asi 10 až 14 let, které přímo na místě měnily zadání a programovaly další činnosti. Tato firma velmi dobře zvládá propojení elektroniky, elektrického pohonu a ovládání, mechanických převodů a pneumatických a hydraulických dílů (obr. 47 a obr. 48). Právě pneumatické součástky ve stavebnicích patří mezi specialitu firmy Fischertechnik.

V sortimentu téže firmy najdeme i různé hračky a vozítka na solární pohon, což je především v zemích EU jednak oblíbený žánr, jednak i silně podporované nejen ekologickými organizacemi, ale i státem. Trend, který směřuje k výchově dětí a především v jejich řadách k velké popularizaci obnovitelných zdrojů energie, zejména solární, nestojí nyní mnoho peněz, ale v budoucnosti přinese nepochybně značný užitek. To je jedna z věcí, které by se v našich podmínkách daly dělat také, ale aktivity v tomto směru jsou velmi vzácné.



Obr. 46. Lego Mindstorms Robotics Invention System 2.0

Závěr

Uvedené příklady úloh, jejich řešení i naznačené rozšíření, mohou sloužit jako výchozí materiál pro mnoho dalších pokusů. Je možné z nich i vybrat jen určité prvky, které jsou zvládnutelné pro menší děti, nebo se naopak orientovat především na logické a funkční nadstavby programů.

Je více oblastí, které zde nebyly vůbec zmíněny. Např. řízení krokových motorů, které mají pro ovládání velmi zajímavou vlastnost. Řízením můžeme přesně regulovat rychlost a počet otáček, počítač tedy stále ví, v jaké poloze se hřídel motoru nachází. Na druhou stranu využitelná síla je proti modelářským servům jen velmi malá. Krokové motory lze snadno získat z vyřazených mechanik „velkých" disketových jednotek ze starých PC.

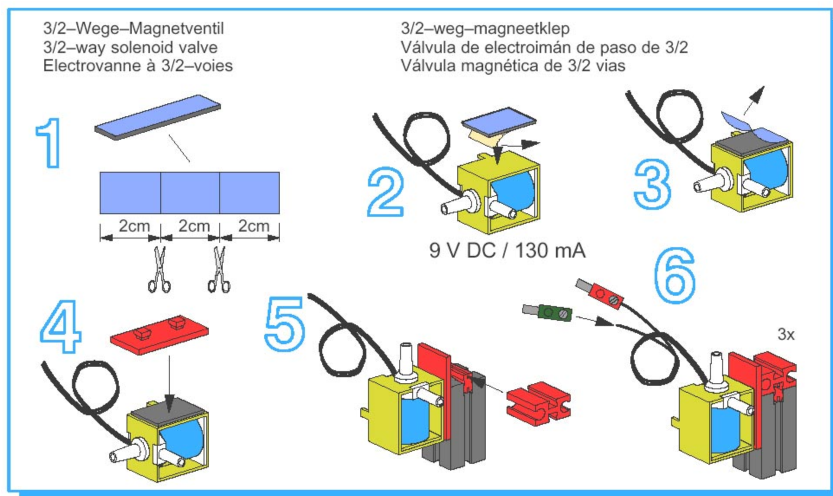
Další, technicky jednoduchou a pro děti velmi zábavnou oblastí je konstrukce nejrůznějších zabezpečovacích a hlídacích zařízení s více čidly. Příklady, kdy se jedna skupina snaží vymyslet zajištění předmětu - hračky - proti odcizení a druhá naopak přijít na to, jak zabezpečovací zařízení obejít a nezpůsobit poplach, nutí obě skupiny k přemýšlení a vynalézavosti, jejímž výsledkem jsou často naprosto nečekaná řešení.

Zabývali jsme se pouze příklady, k nimž stačil uvedený přípravek Interface PC. Pokud je potřebný počet výstupů nebo i vstupů podstatně větší (až do řádu stovek), můžeme data vysílat sériově a dekódovat je posuvným registrem. Pro aplikace, kde je spojení počítače a řízeného předmětu kabelem příliš omezující, se dají nasadit již zmíněné homologované moduly vysílače a přijímače na 433 MHz nebo přenos IR světlem. Data se přenášejí v sériové formě a za přijímačem se dekódují jednočipovým procesorem.

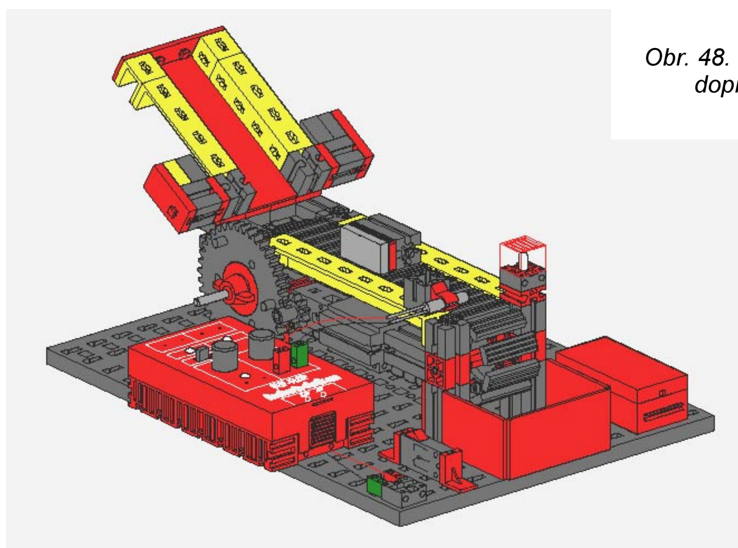
Mechanické ani elektronické řešení není však ve většině případů to podstatné. Je nutné mít především na

zřeteli, že smyslem úloh je ukázat dětem všech věkových skupin, že počítač není jen na počítání, ani jen na hraní her, psaní textů nebo jako terminál připojení k internetu, ale že je to také prostředek k řízení a ovládání věcí kolem nás. A že právě tato činnost je krásná tím, že vyžaduje mnohem více tvůrčího přístupu a myšlení.

Na případné dotazy odpoví autor tohoto článku elektronickou poštou (E-mail) na adrese:
michal.cerny@volny.cz



Obr. 47. *Elektrický ventil firmy Fischertechnik*



Obr. 48. *Sestavený dopravník*

Literatura

[1] *Rambousek, V. a kolektiv*: Práce s počítačem, Praktické činnosti pro 6. až 9. ročník. Fortuna, Praha 1997.

[2] *Pecinovský, R.; Vácha, J.*: SGP Baltík: učebnice programování nejen pro děti.

[3] *Kainka, B.; Berndt, H. J.*: Využití rozhraní PC pod Windows. HEL, Ostrava 2000.

[4] *Kroczek, J.; Steinbauer, M.*: Delfín. Praktická elektronika A Radio 7/1996, s. 9 až 11.

[5] Zapojení konektorů v PC. Praktická elektronika A Radio 1/2001, s. 36 až 37, Praktická elektronika A Radio 2/2001, s. 36 až 37.

[6] *Šedý, V.*: Rozeberte si PC. BEN, Praha 2000.

[7] *Wojciechowski, J.*: Amatérske elektronické modely. ALFA, Bratislava 1975 (originál z roku 1966).

[8] *Steeman, H.*: Robotic Invention System. Elektor 4/2000.

Nová radioamatérská učebnice

V měsíci říjnu vyšla v našem vydavatelství kniha Českého radioklubu s názvem



Požadavky ke zkouškám operátorů amatérských rádiových stanic

Je to již páté vydání; této učebnice bylo v uplynulých létech prodáno přes 10 000 výtisků. Nynější vydání je opět aktualizováno, neboť za poslední dva roky se podstatně změnily předpisy, týkající se radioamatérského vysílání. Autory učebnice jsou Ing. J. Kadlčák a Ing. M. Prostecký a je určena především zájemcům o složení zkoušek nutných k získání koncese (licence) na radioamatérskou vysílací stanici, neboť v ní najdou odpovědi na všechny otázky, kladené u zkoušek. Vzhledem ke svému obsahu je však stejně vhodná jako učebnice základů radiotechniky. Je zpracována velmi přehledně; přibližně polovina knihy je věnována radioamatérským předpisům a radioamatérskému provozu od Mezinárodního radiokomunikačního řádu až po naše nejnovější radioamatérské provozní a technické podmínky (vyhláška MDS č. 201/2000 Sb.). Druhá polovina knihy podrobně probírá základy radiotechniky ve čtrnácti kapitolách (např. Základy elektrotechniky, Základy rádiového přenosu, Zdroje elektrické energie, Polovodiče, Základní elektronické obvody, Antény atd.).

Kniha je v brožované vazbě, formátu A5, má 250 stran s 240 obrázky. Je k dostání v Českém radioklubu, v prodejnách s radioamatérským zbožím a technickou literaturou nebo si ji můžete objednat a koupit (cena 160 Kč + poštovné a balné) přímo v našem vydavatelství:

AMARO, Radlická 2, 150 00 Praha 5, tel./fax: (02) 57 31 73 13, (02) 57 31 73 12, E-mail: pe@aradio.cz